68' MICRO JOURNAL

OS-9 Atari Amiga Mac S-50

6800 6809 68008 68000 68010 68020 68030

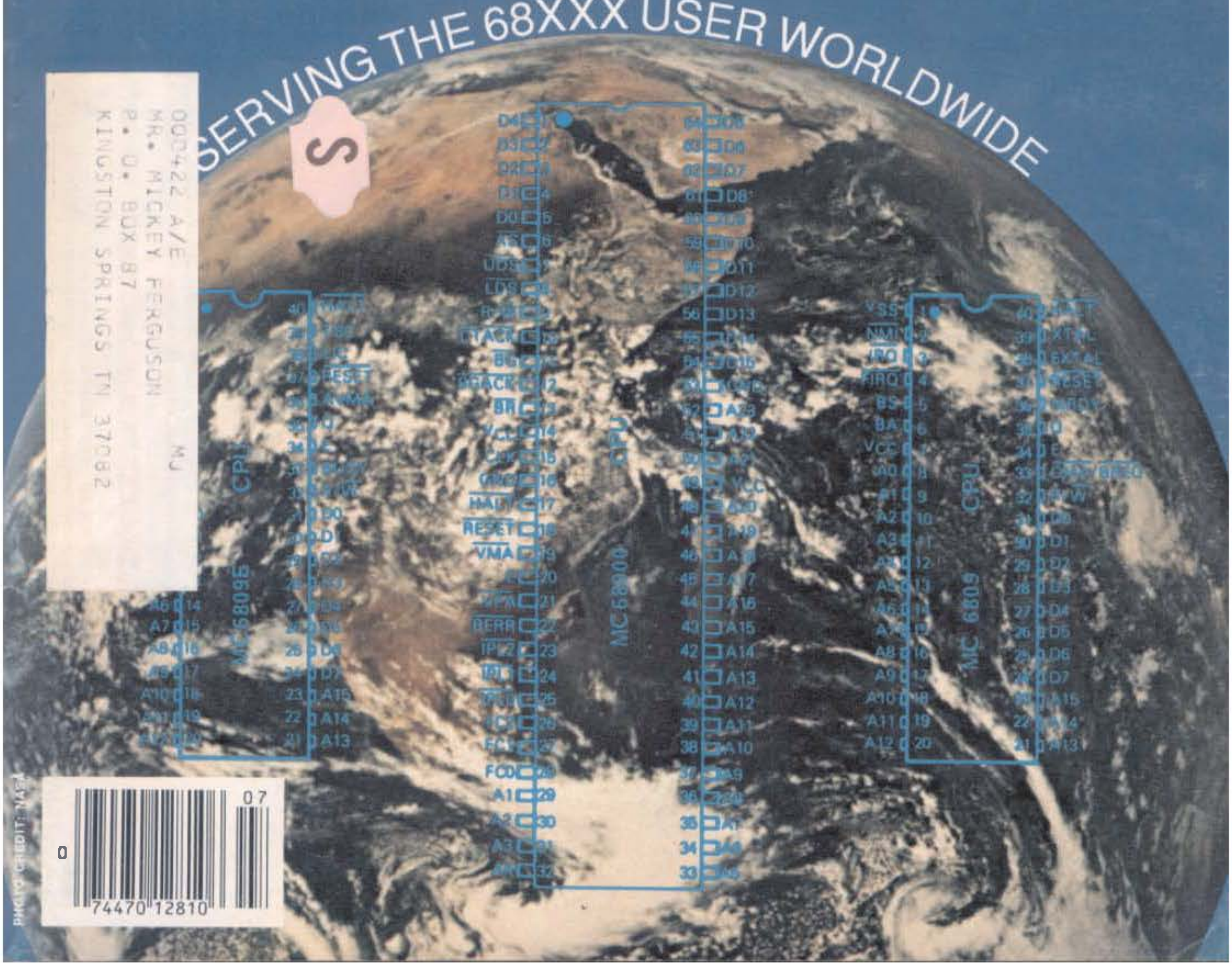
The Magazine for Motorola CPU Devices *For Over a Decade!*

This Issue:

OS-9 SK*DOS Atari Amiga FLEX Macintosh *A User Contributor Journal* And Lots More!

VOLUME X ISSUE VII ● Devoted to the 68XXX User ● July 1988

The Grandfather of "DeskTop Publishing™"



WHO DO YOU CALL WHEN YOUR DEBUGGER WON'T DEBUG?

The problem with most real-time operating systems is simple, they're not an integrated solution. You end up dealing with a multitude of suppliers for languages, compilers, debuggers and other important development tools. And when something does go wrong, it can be a frustrating experience trying to straighten out the mess.

Modularity Lets YOU Choose Just What You Need.

The modular design of OS-9 allows our Operating System to adapt as your requirements change. OS-9 can support a complete spectrum of applications — from embedded ROM-based code in board-level products all the way up to large-scale systems.

Why Not Try the Microware One-Stop Total Solution?

Microware's OS-9 Real-Time Operating System is a total integrated software system, not just a kernel. We offer an extensive set of development tools, languages, I/O and Kernel options. ***And this total integrated solution is entirely designed, built and supported by the same expert Microware team.***

Microware is a registered trademark of Microware Systems Corporation.
OS-9 is a trademark of Microware.
UNIX is a trademark of AT&T.
VAX is a trademark of DEC.

The OS-9 Success Kit

A Total Integrated Solution for Your Next Project

Development Tools:

C Source Level Debugger
Symbolic Debugger
System State Debugger
uMACS Text Editor
Electronic Mail
Communications
Super Shell

Kernel Options:

MMU (Security Protection) Support
Math Coprocessor Support

Languages:

C*
Basic
Pascal
Fortran
Ada**
Assembler*

I/O Options:

SCSI, SASI & SMD Disks
3-, 5-, 8-inch Diskettes
Magnetic Tape
Ethernet - TCP/IP
Arcnet - OS-9/Net

*Resident or UNIX versions available
**VAX hosted

Support is Part of the Package.

Microware is proudly setting the industry's standard for customer support. You'll find professional and comprehensive technical documentation and a Customer Hotline staffed by courteous and authoritative software engineers.

So stop messing with simple kernels and independent suppliers. Call Microware today and find out more about the "One-Stop Integrated Solution" with OS-9!

microware® OS-9

Microware Systems Corporation
1900 N.W. 114th Street
Des Moines, Iowa 50322
Phone: 515/224-1929

Western Regional Office
4401 Great America Parkway
Santa Clara, California 95054
Phone: 408/980-0201

Microware Japan Ltd.
41-19 Honcho 4-Chome
Funabashi City
Chiba 273, Japan
Phone: 0474 (22) 1747

GMX MICRO-20 and TWINGLE-20 PRICE LIST

All versions include 1 SAB Board

	MICRO-20 with 1MB RAM	MICRO-20 with 2MB RAM	TWINGLE-20 with 4MB RAM
12.5 MHz	1855.00	2155.00	3855.00
16.67 MHz	2185.00	2485.00	4185.00
20 MHz	2585.00	2885.00	4785.00

OPTIONAL PARTS AND ACCESSORIES

68881 12.5MHz Floating Point Coprocessor	$ 165.00
68881 16.67MHz Floating Point Coprocessor	$ 225.00
68881 20MHz Floating Point Coprocessor	$ 345.00
MOTOROLA 68020 USERS MANUAL	$ 18.00
MOTOROLA 68030 USERS MANUAL	$ 18.00
MOTOROLA 68881 USERS MANUAL	$ 18.00

SBC ACCESSORY PACKAGE (M20-AP) $1399.00

The package includes a PC-style cabinet with a custom backpanel, a 25 Megabyte (unformatted) hard disk and controller, a floppy disk drive, a 150 watt power supply, cooling fan, panel mounted reset and abort switches, and all necessary internal cabling. (For use with SAB—9D serial connectors only.)

2nd 5"80 FLOPPY & CABLES FOR M20-AP, ADD	$ 250.00
SECOND 25MB HARD DISK & CABLES, ADD	$ 780.00
TO SUBSTITUTE 50MB HD FOR 25MB HD, ADD	$ 290.00
TO SUBSTITUTE 80MB HD FOR 25MB HD, ADD	$1500.00
TO SUBSTITUTE 155MB FOR 25MB HD, ADD	$2100.00
60MB TEAC STREAMER WITH ONE TAPE	$ 690.00
PKG. OF 5 TEAC TAPES	$ 112.50
CUSTOM BACK PANEL PLATE (BPP-PC)	$ 44.00

I/O EXPANSION BOARDS

16 PORT SERIAL BOARD ONLY (SBC-16S) .... $ 335.00

The SBC-16S extends the I/O capabilities of the GMX Micro-20 68020 Single-board Computer by adding sixteen asynchronous serial I/O ports. By using two SBC-16S boards, a total of thirty-six serial ports are possible.

RS232 ADAPTER (SAB-25, SAB-9D or SAB-8M) $165.00

The board provides level-shifting between TTL level and standard RS-232 signal levels for up to 4 serial I/O ports.

60 LINE PARALLEL I/O BOARD (SBC-60P) $398.00

The GMX SBC-60P uses three 68230 Parallel interface/Timers (PI/Ts) to provide up to forty-eight parallel I/O lines. The I/O lines are buffered in six groups of eight lines each, with separate buffer direction control for each group. Buffer direction can be fixed by hardware jumpers, or can be software programmable for bidirectional applications.

PROTOTYPING BOARD (SBC-WW) $75.00

The SBC-WW provides a means of developing and testing custom I/O interface designs for the GMX Micro-20 68020 Single-board Computer. The board provides areas for both DIP (Dual Inline Package) and PGA (Pin Grid Array) devices, and a pre-wired memory area for up to 512K bytes of dynamic RAM.

I/O BUS ADAPTER (SBC-BA) $195.00

The SBC-BA provides an interface between the GMX Micro-20 68020 Single-board Computer and the Motorola Input/Output Channel (I/O bus). With the I/O bus, up to sixteen off-the-shelf or custom peripheral devices (I/O modules) can be connected to the GMX Micro-20.

ARCNET LAN board w/o Software (SBC-AN) $475.00

The SBC-AN provides an interface between the GMX Micro-20 68020 Single-board Computer and the ARCNET modified token-passing Local Area Network (LAN) originally developed by Datapoint Corp. The ARCNET is a baseband network with a data transmission rate of 2.5 Megabits/second. The standard transmission media is a single 93 ohm RG-62/U coaxial cable. Fiber optic versions are available as an option.

OS9 LAN Software Drivers for SBC-AN 120.00

GMX MICRO-20 SOFTWARE

020 BUG UPDATE — PROMS & MANUAL $150.00

THESE 68020 OPERATING SYSTEMS ARE PRICED WHEN PURCHASED WITH THE MICRO-20. PLEASE ADD $150.00 IF PURCHASED LATER FOR THE UPDATED PROMS AND MANUALS. ALL SHIPPED STANDARD ON 5¼" DISKS 3½" OPTIONAL IF SPECIFIED.

OS9/68020 PROFESSIONAL PAK $850.00

Includes O.S., "C", uMACS EDITOR, ASSEMBLER, DEBUGGER, development utilities, 68881 support.

OS9/68020 PERSONAL PAK .... $ 400.00

Personal OS-9 systems require a GMX Micro-20 development system running Professional OS-9/68020 for initial configuration.

BASIC (included in PERSONAL PAK)	$ 200.00
C COMPILER (included in PROFESSIONAL PAK)	$ 750.00
PASCAL COMPILER	$ 500.00

UNIFLEX

UniFLEX (for Micro-20)	$ 400.00
UniFLEX WITH REAL-TIME ENHANCEMENTS	$ 800.00
UniFLEX VM (for TWINGLE-20)	$ 600.00
UniFLEX VM REAL-TIME ENHANCEMENTS	$1000.00

Other Software for UniFLEX

UniFLEX BASIC W/PRECOMPILER	$ 300.00
UniFLEX C COMPILER	$ 350.00
UniFLEX COBOL COMPILER	$ 750.00
UniFLEX SCREEN EDITOR	$ 150.00
UniFLEX TEXT PROCESSOR	$ 200.00
UniFLEX SORT/MERGE PACKAGE	$ 200.00
UniFLEX VSAM MODULE	$ 100.00
UniFLEX UTILITIES PACKAGE I	$ 200.00
UniFLEX PARTIAL SOURCE LICENSE	$1000.00

GMX EXCLUSIVE VERSIONS, CUSTOMIZED FOR THE MICRO-20, OF THE BELOW LANGUAGES AND SOFTWARE ARE ALSO AVAILABLE FROM GMX.

ABSOFT FORTRAN (UniFLEX)	$1500.00
SCULPTOR (specify UniFLEX or OS9)	$ 995.00
FORTH (OS9)	$ 595.00
DYNACALC (specify UniFLEX or OS9)	$ 300.00

GMX DOES NOT GUARANTEE PERFORMANCE OF ANY GMX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

ALL PRICES ARE F.O.B. CHICAGO IN U.S. FUNDS

GMX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

CONTACT GMX FOR MORE INFORMATION ON THE ABOVE PRODUCTS

GMX STILL SELLS GIMIX S50 BUS SYSTEMS, BOARDS & PARTS. CONTACT GMX FOR COMPLETE PRICE LIST.

GMX 1337 W. 37th Place, Chicago, IL 60609 (312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Publisher
Don Williams Sr.

Executive Editor
Larry Williams

Production Manager
Tom Williams

Office Manager
Joyce Williams

Subscriptions
Stacy Power

Contributing & Associate Editors

Ron Anderson
Ron Voigts
Doug Lurie
Ed Law

Dr. E.M. "Bud" Pass
Art Weller
Dr. Theo Elbert
& Hundreds More of Us

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.

"Over a Decade of Service"

"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN. and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of gripe as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9
68XXX
Systems

service
For Over a Decade!
1,200 Titles +

A Division of
CPI

S.E.
Media
5900 Cassandra Smith Rd.
Hixson, Tn 37343
Telephone 615 842-6809
Telex 510 600-6630

6800-6809-68008-68000-68010-68020

PAT FROM S. E. MEDIA -- A FULL FEATURED SCREEN ORIENTED TEXT EDITOR with all the best of PIE. For those who swore by and loved PIE, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included.** Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO

PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

MUSTANG -020 Super SBC ™

DATA-COMP Proudly Presents the First Under $4300 "SUPER MICRO" See other advertising (backcover) for economy system (68008) - under $2400 complete!

™ CPI
Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $4300, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 32serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $2500. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

Available 12.5 - 25 Mhz systems, call for special prises

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based debugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

Data-Comp Division
CPI A Decade of Quality Service*
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

Mustang-020 Mustang-08 Beochmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	5.1	3.2
68008 OS-9 68K 8 Mhz	18.0	9.0
68000 OS-9 68K 10 Mhz	6.5	4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	*9.8*	*6.3*
MUSTANG-020 68020 OS-9 68K 16 Mhz	*2.2*	*0.88*
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	*1.8*	*1.22*

```
Main()
            {
                  register long i;
                  for (i=0; i < 999999; ++i);
            }
```

Estimated MIPS - MUSTANG-020 ..... 4.5 MIPS,

Burst to 8 - 10 MIPS: Motorola Specs

Mustang Specifications

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
Tered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

OS-9

OS-9 Professional Ver *includes C Compiler	$850.00
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Fortran 77	750.00
Microware Pascal	500.00
Omegasoft Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Sculptor+ (see below)	995.00
Standard MUSTANG-020™ shipped 12.5 Mhz.	
Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232	335.00
Requires 1 or 2 Adapter Cards below RS232 Adapter Each card supports 4 additional ser. ports (total of 36 serial ports supported)	165.00
60 line Parallel I/O card Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.	398.00
Prototype Board areas for both dip and PGA devices & a pre-wired memory area up to 512K DRAM.	75.00
SBC-AN interface between the system and ARCNET modified token-passing LAN, fiber optics optional - call.	475.00
LAN software drivers	120.00
Expansion for Motorola I/O Channel Modules	$195.00

Special for complete MUSTANG-020™ system buyers - Sculptor+ $695.00. SAVE $300.00

Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

The PRO!

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a must.

Only the "PRO" Version of OS-9 Supported!

This is *HEAVY DUTY* Country!

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information

Price List:

Mustang-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy DS/DD		$269.95
Floppy Cable		$39.95
OS-9 68K Professional Version		$850.00
C Compiler ($500 Value)		N/C
Winchester Cablo		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk Controller		$395.00
Shipping USA UPS		$20.00
UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00
Note all 68881 chips work with 20 Mhz Sys		
	Total:	$5299.80

Save $1000.00
Complete
25 Mbyte HD System
$4299.80
85Mbyte HD System
$5748.80

Note: Only Professional OS-9 Now Available (68020 Version)
Includes ($500) C Compiler - 68020 & 68881 Supported -
For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

Data-Comp Division
A Decade of Quality Service"
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

C

The C Programmers Reference Source. Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

This chapter begins the discussion of dbug, a C debugging package. It is a useful tool for debugging and testing C programs. It was developed by Fred Fish, who placed it into the public domain. Much of the coverage following is based on the documentation provided by Fred, who is also responsible for the floating-point library discussed in previous chapters. The C code for the dbug package and for extensions to it appear in subsequent chapters.

FRED FISH'S DBUG PACKAGE

All of the features of the dbug package can be enabled or disabled dynamically at execution time. They may also be disabled statically at compilation time. This means that production programs will run normally when debugging is not enabled, and multiple versions of a program for production and for testing need not be maintained.

Many of the debugging actions easily accomplished with conventional debugging tools, such as symbolic debuggers, are difficult or impossible to accomplish with this package, and vice versa. Thus, the dbug package should not be thought of as a replacement or substitute for other debugging tools, but as a useful addition to the familiar set of tools, if they are available on a given system.

Almost every non-trivial program development and debugging environment provides some sort of debugging facility. Usually this takes the form of a program which is capable of controlling execution of other programs and examining the internal state of other executing programs.

Such programs are examples of external debuggers since the debugger is not part of the executing program. Examples of this class of debugger program include the adb and sdb debuggers provided with UNIX, and, at a much lower level, the debug debuggers provided with MSDOS, FLEX, UNIFLEX, and OS-9.

One of the problems associated with developing programs in an environment with good external debuggers is that developed programs tend to have little or no internal instrumentation.

This is usually not a problem for the developer since they are, or at least should be, intimately familiar with the internal organization, data structures, and control flow of the program being debugged.

It is a serious problem for maintenance programmers, who are unlikely to have such familiarity with the program being maintained, modified, or ported to another environment.

It is also a problem, even for the developer, when the program is moved to an environment with a primitive or unfamiliar debugger, or even no debugger.

The dbug package is an example of an internal debugger, because it requires internal instrumentation of a program. Its usage does not depend on any special capabilities of the execution environment. It is always available and will execute in any environment in which that the program itself will execute. Since dbug is a complete package with a specific user interface, all programs which use it will be provided with similar debugging capabilities.

The dbug package imposes only a slight speed overhead on executing programs, typically much less than 10 percent, and a modest size overhead, typically 10 to 20 percent. By defining a C preprocessor symbol (DBUG_OFF), both of these can be reduced to zero, with no modifications required to the source code.

DBUG CAPABILITIES

Following is a summary of the capabilities of the dbug package. Each capability may be enabled or disabled at the time a program is invoked by specifying the appropriate command line arguments.

Execution trace showing function-level control-flow in a graphical manner using indentation to indicate nesting depth.

Output the values of all, or any subset of, internal variables.

Limit actions to a specific set of named functions.

Limit function trace to a specified nesting depth.

Label each output line with source file name and line number.

Label each output line with name of current process.

Push or pop internal debugging state to allow execution with built-in debugging defaults.

Redirect the debug output stream to standard output (stdout) or a named file. The default output stream is standard error (stderr).

PRIMITIVE DEBUGGING TECHNIQUES

Internal instrumentation is already a familiar concept to most C programmers, since it is usually the first debugging technique learned. Typically, print and/or assertion statements are inserted into the source code at strategic locations, the code is recompiled and executed, and the resulting output is examined in an attempt to determine where the problem is.

The procedure is iterative, with each iteration yielding more and more output, and hopefully the source of the problem is discovered before the output becomes too large to deal with or previously inserted statements need to be removed. Following is a trivial example of this type of primitive debugging technique.

```
#include <stdio.h>

main (argc, argv)
int argc;
char *argv[];
{
    int i;

    for (i = 0; i < argc; ++i)
        printf ("argv[i] = %s\n", argv[i]);
    :
    :
    printf ("argc = %d\n", argc);
    :
    :
    printf ("== done ==\n");
}
```

Usually after several iterations, the problem will be isolated and corrected.

At this point, the newly-inserted debugging print statements must be eliminated or disabled to prevent them from corrupting the normal program output.

One obvious solution is to simply delete the debugging statements. Beginners usually do this a few times until they have to repeat the entire process every time a new bug is discovered or extensive modifications must be performed on the program.

Another obvious solution is to somehow disable the output, either through the preprocessor /* - */ comment facility, the preprocessor #ifdef - #endif exclusion facility, or the creation of a debug variable to be switched on or off as needed.

Following is an example of the use of all three techniques.

```
#include <stdio.h>

int debug;

main (argc, argv)
int argc;
char *argv[];
{
    int i;

    /*
    for (i = 0; i < argc; ++i)
        printf ("argv[i] = %s\n", argv[i]);
    */
    :
    :
    if (debug)
        printf ("argc = %d\n", argc);
    :
    :
#ifdef DEBUG
    printf ("== done ==\n");
#endif
}
```

Each technique has its advantages and disadvantages with respect to dynamic versus static optinal usage, source code overhead, recompilation requirements, ease of use, program readability, etc. Overuse of the preprocessor solution quickly leads to major problems with source code readability and maintainability when a multitude of preprocessor symbols must be #define'ed or #undef'ed based on specific types of debugging output required.

FUNCTION TRACE EXAMPLE

The capabilities of Fred Fish's dbug package are demonstrated thru the use of a simple program which computes the factorial of a number. In order to better demonstrate the function trace mechanism, this program is implemented recursively.

Following is the main function for this factorial program.

```
    #include <stdio.h>
    #include "dbug.h"

    main (argc, argv)
    int argc;
    char *argv[];
    {
        register int ix;
        register int result;
        extern int atoi ();
        extern int factorial ();

        DBUG_ENTER ("main");
        DBUG_PROCESS (argv[0]);
        for (ix = 1; ix < argc && argv[ix][0] == '-';
ix++)
        {
            switch (argv[ix][1])
            {
            case '#':
                DBUG_PUSH (&(argv[ix][2]));
                break;
            }
        }
        for ( ; ix < argc; ix++)
        {
            DBUG_PRINT ("args", ("argv[%d] = %s", ix,
argv[ix]));
            result = factorial (atoi (argv[ix]));
            printf ("%d\n", result);
        }
        DBUG_RETURN (0);
    }
```

The main function is responsible for processing any command line option arguments and then computing and printing the factorial of each non-option argument.

The debugging functions are implemented via preprocessor commands. This does not detract from the readability of the code and makes disabling all debug compilation trivial, as the existence of preprocessor symbol DBUG_OFF causes the dbug commands to be nullified.

The header file dbug.h must be included from the local header file directory whenever the dbug package is to be used. This file contains all the definitions for the debugging commands, which have the form DBUG_XX...XX.

Following are the dbug commands used in this example program:

The DBUG_ENTER command indicates that a function has been entered. It must be the first executable line in a function, after all declarations and before any other executable lines.

The DBUG_PROCESS command is used only once per program to indicate the name under which the program was invoked.

The DBUG_PUSH command modifies the current debugging state by saving the previous state and setting a new state based on the control string passed as its argument.

The DBUG_PRINT command prints the values of each of its argument.

The DBUG_RETURN command indicates that the end of the function has been encountered and returns a value to the calling function.

One advantage of using the dbug package is that it encourages structured coding with only one entry and one exit point in each function. Multiple exit points, such as early returns to escape a loop, may be used, but each such point requires the use of an appropriate DBUG_RETURN or DBUG_VOID_RETURN command.

To invoke the debugger, the factorial program is invoked with a command line of the following form:

```
factorial -#d:t 1 2 3
```

The main function recognizes the "-#d:t" string as a debugging control string, and passes the debugging arguments ("d:t") to the dbug runtime support routines via the DBUG_PUSH command. This particular string enables output from the DBUG_PRINT command with the 'd' flag and enables function tracing with the 't' flag. The factorial function is then called three times, with the arguments "1", "2", and "3". Note that DBUG_PRINT requires exactly two arguments, with the second argument (a format string and list of printable values) enclosed in parentheses.

Debug control strings consist of a header, the "-#", followed by a colon separated list of debugging arguments. Each debugging argument is a single character flag followed by an optional comma separated list of arguments specific to the given flag.

Some examples are as follows:

```
-#d:t:o -#d,in,out:f,main:F:L
```

Note that previously-enabled debugging actions can be disabled by the control string "-#".

The definition of the factorial function, "N!", is given by the following formula:

N! = N * (N - 1) * ... 2 * 1

Following is the factorial function which implements this algorithm recursively. Note that this is not necessarily the best way to calculate factorials and that any error conditions are ignored.

```
    #include <stdio.h>
    #include "dbug.h"

    int factorial (value)
    register int value;
    {
        DBUG_ENTER ("factorial");
        DBUG_PRINT ("find", ("find %d factorial",
value));
        if (value > 1)
            value *= factorial (value - 1);
        DBUG_PRINT ("result", ("result is %d",
value));
        DBUG_RETURN (value);
    }
```

To build the factorial program on a UNIX system, compile and link with the following command:

```
cc -o factorial main.c factorial.c dbug.c
```

Executing the factorial program with a command of the following form:

```
factorial 1 2 3 4 5
```

generates the output shown in the following list:

```
1
2
6
24
120
```

Function level tracing is enabled by passing the debugger the 't' flag in the debug control string.

Following is the output resulting from the following command:

```
factorial -#t:o 3 2

|  >factorial
|  |  >factorial
|  |  <factorial
|  <factorial
2
|  >factorial
|  |  >factorial
|  |  |  >factorial
|  |  |  <factorial
|  |  <factorial
|  <factorial
6
<main
```

Each entry to or return from a function is indicated by '>' for the entry point and '<' for the exit point, connected by vertical bars to allow matching points to be easily found when separated.

This trace output indicates that there was an initial call to factorial from main (to compute 2!), followed by a single recursive call to factorial to compute 1!. The main program then output the result for 2! and called the factorial function again with the second argument, 3. Factorial called itself recursively to compute 2! and 1!, then returned control to main, which output the value for 3! and exited.

Note that there is no matching entry point "main>" for the return point "<main" because at

the time the DBUG_ENTER command was reached in main, tracing was not enabled yet. It was only after the command DBUG_PUSH was executing that tracing became enabled. This implies that the argument list should be processed as early as possible since all code preceding the first call to DBUG_PUSH is essentially invisible to dbug. This can be circumvented by inserting DBUG_PUSH(argv[1]) immediately after the DBUG_ENTER("main") command.

The trace output is normally produced on the standard error file.

Since the factorial program prints its result on the standard output, there is the possibility of the output of the program being scrambled if the two streams are not synchronized.

Thus the debugger is told to write its output on the standard output instead, via the 'o' flag character. No 'o' implies the default (standard error), a 'o' with no arguments means standard output, and a 'o' with an argument means use the named file.

For example, the following command:

```
factorial -#t:o,logfile 3 2
```

would place the trace output in "logfile". Because of implementation details, programs usually run faster when writing to stdout rather than to stderr on many operating systems, though this is not a prime consideration in this example.

DBUG_PRINT COMMAND

The mechanism used to produce printed output is the DBUG_PRINT command.

To allow selection of output from specific command, the first argument to every DBUG_PRINT command is a dbug keyword. When this keyword appears in the argument list of the 'd' flag in a debug control string, as in "-#d,keyword1,keyword2,...:t", output from the corresponding command is enabled. The default when there is no 'd' flag in the control string is to enable output from all DBUG_PRINT command.

Typically, a program will be run once, with no keywords specified, to determine what keywords are significant for the current problem (the keywords are printed in the command output line). Then the program will be run again, with the desired keywords, to examine only specific areas of interest.

The second argument to a DBUG_PRINT command is a standard printf style format string and one or more arguments to print, all enclosed in parentheses, so that they collectively become a single command argument. This is how variable numbers of printf arguments are supported. Also note that no explicit newline is required at the end of the format string. As a matter of style, two or three small DBUG_PRINT command are preferable to a single command with a huge format string.

Below is the output for default tracing and debugging, using the following command line:

```
factorial -#d:t:o 3

|   args: argv[2] = 3
|   >factorial
|   |   find: find 3 factorial
|   |   >factorial
|   |   |   find: find 2 factorial
|   |   |   >factorial
|   |   |   |   find: find 1 factorial
|   |   |   |   result: result is 1
|   |   |   <factorial
|   |   |   result: result is 2
|   |   <factorial
|   |   result: result is 6
|   <factorial
6
<main
```

The output from the DBUG_PRINT command is indented to match the trace output for the function in which the command occurs. When debugging is enabled, but not trace, the output starts at the left margin, without indentation.

To demonstrate selection of specific command for output, below is the result when the factorial program is invoked with the following debug control string:

```
-#d,result:o

factorial: result: result is 1
factorial: result: result is 2
factorial: result: result is 6
factorial: result: result is 24
24
```

It is sometimes desirable to restrict debugging and trace actions to a specific function or list of functions. This may be accomplished with the 'f' flag character in the debug control string. The 'F' flag enables printing of the source file name and the 'L' flag enables printing of the source file line number.

Below is the output of the factorial program when run with the following control string:

```
-#d:f,factorial:F:L:o

  factorial.c:    8: factorial: find: find 3 factorial
  factorial.c:    8: factorial: find: find 2 factorial
  factorial.c:    8: factorial: find: find 1 factorial
  factorial.c:   11: factorial: result: result is 1
  factorial.c:   11: factorial: result: result is 2
  factorial.c:   11: factorial: result: result is 6
6
```

The output shows that the "find" command is in file "factorial.c" at source line 8 and the "result" command is in the same file at source line 11.

SUMMARY OF DBUG COMMANDS AND CAPABILITIES

The following summarizes the usage of all commands defined in the dbug package. The command definitions are found in the user include file dbug.h from the standard include directory.

DBUG_ENTER(char *)

The DBUG_ENTER command is used to tell the runtime support module the name of the function being entered. It must precede all other executable lines in the function just entered, and must be placed after all local declarations. Each DBUG_ENTER command must have a matching DBUG_RETURN or DBUG_VOID_RETURN command at the function exit point. DBUG_ENTER command used without a matching DBUG_RETURN or DBUG_VOID_RETURN command will cause warning messages from the dbug package runtime support module.

DBUG_RETURN(value) and DBUG_VOID_RETURN

The DBUG_RETURN and DBUG_VOID_RETURN commands are used at each exit point of a function containing a DBUG_ENTER command at the entry point. The argument of the DBUG_RETURN command is the value to return. Functions which return no value (void) should use the DBUG_VOID_RETURN command. It is an error to have a DBUG_RETURN or DBUG_VOID_RETURN command in a function which has no matching DBUG_ENTER command.

DBUG_PROCESS(char *)

The DBUG_PROCESS command is used to name the program being executed. A typical argument for this command is "argv[0]".

DBUG_PUSH(char *)

The DBUP_PUSH command sets a new debugging state by pushing the current debugging state onto an internal stack and setting up the new state using the debug control string passed as the command argument. The most common usage is to set the state specified by a debug control string retrieved from the argument list. Note that the leading "-#" in a debug control string specified as a command line argument must not be passed as part of the command argument. The proper usage is to pass a pointer to the first character after the "-#" string.

DBUG_POP()

The DBUG_POP command restores the previous debugging state by popping the state stack. Attempting to pop more states than pushed will be ignored and no warning will be given.

FILE *DBUG_FILE

The DBUG_FILE command is used to do explicit I/O on the debug output stream. It is used in the same manner as the symbols "stdout" and "stderr" in the standard I/O package.

DBUG_EXECUTE(char *, fcn())

The DBUG_EXECUTE command is used to execute any arbitrary C code. The first argument is the debug keyword, used to trigger execution of the code specified as the second argument. This command must be used cautiously because, like the DBUG_PRINT command, it is automatically selected by default whenever the 'd' flag has no argument list (i.e., a "-#d:t" control string).

DBUG_PRINT(char *,(char *))

The DBUG_PRINT command is used to do printing via the fprintf library function on the current debug stream, DBUG_FILE. The first argument is a debug keyword, the second is a format string and the corresponding argument list. Note that the format string and argument list are all one command argument and must be enclosed in parentheses.

DBUG_SETJMP(env)

The DBUG_SETJMP command is used in place of the setjmp() function to first save the current debugging state and then execute the standard setjmp call. This allows the debugger to restore its state when the DBUG_LONGJMP command is used to invoke the standard longjmp() call. Currently all instances of DBUG_SETJMP must occur within the same function and at the same function nesting level.

DBUG_LONGJMP(env, value)

The DBUG_LONGJMP command is used in place of the longjmp() function to first restore the previous debugging state at the time of the last DBUG_SETJMP and then execute the standard longjmp() call. Currently all DBUG_LONGJMP commands restore the state at the time of the last DBUG_SETJMP. It would be possible to maintain separate DBUG_SETJMP and DBUG_LONGJMP pairs by having the debugging runtime support module use the first argument to differentiate the pairs.

DEBUG CONTROL STRING

The debug control string is used to set the state of the debugging via the DBUG_PUSH command. This section summarizes the currently available debugging options and the flag characters which enable or disable them. Argument lists enclosed in '[' and ']' are optional.

d[,keywords]

Enable output from command with specified keywords. A null list of keywords implies that all keywords are selected.

D[,time]

Delay for specified time after each output line, to let output drain. Time is given in tenths of a second (value of 10 is one second). Default is zero.

f[,functions]

Limit debugging actions to the specified list of functions. A null list of functions implies that all functions are selected.

F

Mark each debugging output line with the name of the source file containing the command causing the output.

g

Turn on machine independent profiling. A profiling data collection file, named dbugmon.out, will be written for postprocessing by the "analyze" program.

L

Mark each debugging output line with the source file line number of the command causing the output.

n

Mark each debugging output line with the current function nesting depth.

N

Sequentially number each debugging output line starting at 1. This is useful for reference purposes when debugging output is interspersed with program output.

o[,file]

Redirect the debugging output stream to the specified file. The default output stream is stderr. A null argument list causes output to be redirected to stdout.

p[,processes]

Limit debugging actions to the specified processes. A null list implies all processes. This is useful for processes which run child processes on multi-tasking systems. Note that each debugging output line can be marked with the name of the current process via the 'P' flag. The process name must match the argument passed to the DBUG_PROCESS command.

P

Mark each debugging output line with the name of the current process on multi-tasking systems. This is most useful with a process which runs child processes that are also being debugged. Note that the parent process must arrange for the debugging control string to be passed to the child processes.

r

Used in conjunction with the DBUG_PUSH command to reset the current indentation level back to zero. Most useful with DBUG_PUSH commands used to temporarily alter the debugging state.

t[,N]

Enable function control flow tracing. The maximum nesting depth is specified by N, and defaults to 200.

HINTS AND TIPS

One of the most useful capabilities of the dbug package is to compare the executions of a given program in two different environments. This is typically done by executing the program in the environment where it behaves properly and saving the debugging output in a reference file. The program is then run with identical inputs in the environment where it misbehaves and the output is again captured in a reference file. The two reference files can then be differentially compared to determine exactly where execution of the two processes diverges.

A related usage is regression testing where the execution of a current version is compared against executions of previous versions. This is most useful when there are only minor changes.

PROBLEM AREAS

The dbug package works best with programs which have "line-oriented" output, such as text processors, general purpose utilities, etc.

It can be interfaced with screen-oriented programs, such as visual editors, by redefining the appropriate dbug commands to call special functions for displaying the debugging results.

Of course, this problem is not encountered if the debugging output is simply dumped into a file for post-execution examination.

Programs which use memory allocation functions other than malloc will often have problems using the standard dbug package without modification. The most common problem is multiply-allocated memory.

DBUG PACKAGE CODE

The actual C code for the dbug package starts in the next chapter.

EOF

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!
6809 - 68020

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL 60139

LOOSE ENDS

This month I decided to spend time wrapping up some loose ends. Loose ends are those little pieces hanging out of a package after you've finished wrapping it. In this case the package is this column and the loose ends are items of information that I find left undone and incorrectly reported. (Gasp! I do sometimes make mistakes.) This month I propose to attack some past material that I believe has some loose ends.

POINTERS

I find that I play loose and free with pointers in C Language from time to time. In many cases the result is an obvious and blatant error. These I catch immediately and don't pass on. But occasionally I get away with creating a pointer error and no one yet has called me on it. (Although one reader, has mentioned this problem, but not necessarily attributing it to me.)

Let me review pointers in C Language. A pointer is a variable that points to a variable. Now that is a fairly simple idea. I don't know of too many other programming languages that use pointers. Pascal does have a type of variable called pointers, but I am not really sure that it is same as the C language type. BASIC09 allows referencing variables by their names. In C, the actual location of the variable can be used. Perhaps a few examples are in order. Here are a few.

```
char name[30];

int *number;

inta,b;

char **lines;
```

The first case, char name[30], is string of 30 bytes. If we just use 'name', we are using its pointer. If we reference it by index,say 'name[5]', we indicate one byte in it. The second cas, int *number, is two bytes long. But it is a pointer to an integer variable. It is not the integer. The third case, int a,b, contains a declaration of two integers, two bytes each. The last one, char **lines, is a pointer to a list of pointers called 'lines'. Again it is only two bytes long.

Knowing how and when to when to use these is the trick. Most times you will want to use a variable and not a pointer to it. Pointers are used when passing a variables location. The can also be used for programming tricks. But whatever the reason, it must be done correctly.

Now let me show you how easy it is to misuse a pointer. The following is how to get the time. This how it is written in the C Manual.

```
#include <time.h>
getime( buffer )
struct sgtbuf *buffer
```

Pretty straight forward, huh? No problems. Declare a buffer as illustrated, pass it to getime and it returns the time. With luck it just might! But this is not how to use this definition.

The definition says that a pointer must be passed, but it is not not correct to declare a pointer. Why? Because no where has the variable been created for the elements of time — year, month, day, hour, minute, and second. Creating a pointer does not provide the variable. Here is a little illustration to prove what I have said.

```
/* Example to show how pointers and
  variables differ */
#include <time.h>
main()
{
      struct sgtbuf *time1;
      struct sgtbuf time2;
      printf("Time1 is %d in size.\n", sizeof( time1 ));
      printf("Time2 is %d in size.\n", sizeof( time2 ));
}
```

I have used two different ways for creating a structure. Now let's try this program.

'Timel' is 2 bytes in size and 'time2' is 6 bytes. The memory requirements for the time buffer must be 6 bytes, so this would indicate that 'time2' should work. But what about 'timel'? It is only a pointer. 'Time1' will be set to a value of $0000. When it is passed to getime(), the 6 bytes will be placed at position $0000.

What occurs at position $0000 can have an effect on what is returned. It happens that on the Coco I have had no problems when this error occurs. On my Smoke Signal Level II, I come back with erroneous results. One reader reported that on his Atari ST system, he experienced crashes. The moral is to use the variable correctly.

This month I have included in listing 1 a program from some time back. This is a replacement for your standard DATE command. This one differs in a few ways. It can print the time in military or standard notation. Also it will print time and date in a greeting fashion.

In this version (Listing 1), I have changed how the structure for the time buffer is declared. This version is correct and creates a real buffer. The early version made the error of creating only a pointer. Take a look at it and see how the buffer is created.

When passing the buffer pointer, a & is added to the front of the variable name. The line

getime(&time);

says to pass the pointer of 'time' to 'getime'. This is the exactly how it should be done, but this time it is done with a real variable declared.

One other change is that I put into this version of DATE a greeting for some holidays. For example, on July 4th, it will say

Happy 4th of July!

I have omitted some holiday. You can add any that you like. Also, you may want to add birthdays, anniversaries and whatever else you like.

SIGNALS, ONE MORE TIME

Last month's topic was signals. I talked about how signals were sent and received. I also presented a little program called DAEMON. It would take over control of the terminal keeping whoever envoked it. from doing anything else. Entering CONTROL-C or CONTROL-E would not stop it. This is where the signal handling came in. The program had a trap to catch incoming signals. It would mock any attempts to halt it. It had only one problem when writing it.

The original version I created had printf() and gets() in it. These two routines are part of the higher order C Library functions. The first version I wrote had a number of problems. Basically, text handled by the higher library functions became messed up and the program ran amok. I took the cowards way out. I rewrote the program using readln() ad writeln(). This solved the problem but did not explain why.

I found an interesting paper written by Kim Kempf called "C Tricks and Treats". The paper was distributed by Microware Systems at one of the annual OS-9 conferences. The paper gave some insight into the problem. (It also inspired the first part of this month's column.) I won't tell all that it contained, but let me give you the run down on what it says regarding the signal handler.

It tells that C Library I/O should not be done in signal handlers. The functions use static storage and therefore not re-enterant. Using such functions can result in unpredictable results.

I did a brief examination of the C library, clib.l. I found that printf() was part of a module called printf_c which had 33 local references. Gets() was part of gets_c which has no local variables. These two modules reference other external symbols that were located in other modules. The results was that there were static variables used when using these functions.

On the other hand, writeln() and readln() were part of a module called io_a. It references no external variables. Therefore no static variables were involved.

I have not gone back and rewritten the program, DAEMON.C. I leave this up to some industrious soul out there. I think that writing it with printf() and gets() would serve no purpose, except to be an exercise. The program would not be anymore transportable since the signal handler is rather specific to the OS-9 system.

Well that concludes another month. I think I have tied up of few loose ends. (And maybe created a few new ones.) Until next time have fun!

LISTING

```
/* ****************************

     Name: Date.c
     Date: 30-MAR-87
     Author: Ron Voigts
```

```
    Compiler: Microware C Compiler

    ****************************

    Version 1.00 Original RDV

    Version 2.00
    Fixed time storage buffer and
    added holiday greeting.   RDV

    ****************************

    Function:
    Prints the date.
    -t = with the time
    -m = time in military notation
    -d = with the day of the week
    -g = with a greeting
    -h = holiday greeting

    **************************** */

#define LEVEL2

#include <stdio.h>
#include <time.h>
#include "getopt.c"

#define TRUE 1
#define FALSE 0

main( argc, argv )
int argc;
char *argv[];
{

struct sgtbuf time;

/* Parameter flags */
int mflag = FALSE; /* Military time */
int tflag = FALSE; /* Print time */
int dflag = FALSE; /* Print the day */
int gflag = FALSE; /* Greeting flag */
int pmflag = TRUE; /* PM flag */
int hflag = FALSE; /* Holiday greeting */

/* Variables used */
  char *option;
  char *optlist="MDTGH";

/* Process the input line */
   optn=1;
   while ( (option=getopt( argc, argv, optlist )) 
!= NULL )
      if ( opterr != 0 )
         dhelp();
      else {
         if ( toupper(option[0]) == 'M' ) mflag = 
TRUE;
         if ( toupper(option[0]) == 'D' ) dflag = 
TRUE;
         if ( toupper(option[0]) == 'T' ) tflag = 
TRUE;
         if ( toupper(option[0]) == 'G' ) gflag = 
TRUE;
         if ( toupper(option[0]) == 'H' ) hflag = 
TRUE;
      }

/* Now get the time */
   getime( &time );

/* Print the greeting */
   if ( gflag ) {
      if ( time.t_hour< 12 )
         printf("Good Morning!\n");
      else if ( time.t_hour<18 )
         printf("Good Afternoon!\n");
      else
         printf("Good Evening!\n");
   }

/* Print the date */
   if ( gflag )
      printf("Today is ");
   pdate( &time );
/* Print the day of the week */
   if ( dflag ) {
      if ( gflag )
         printf("It is ");
      pday( &time );
   }

/* Print the time */
   if ( tflag ) {
      if ( gflag )
         printf("The time is ");
      if ( mflag )
         pmtime( &time );
      else
         ptime( &time, pmflag );
   }

/* Print holiday greeting */
   if ( hflag ) {
      if ( time.t_day==1 && time.t_month==1 )
         printf("Happy New Year!\n");
      if ( time.t_day==14 && time.t_month==2 )
         printf("Happy Valentine's Day!\n");
      if ( time.t_day==17 && time.t_month==3 )
         printf("Happy St. Patrick's Day!\n");
      if ( time.t_day==20 && time.t_month==3 )
         printf("It's the First Day of 
Spring.\n");
      if ( time.t_day==21 && time.t_month==4 )
         printf("It's Armed Forces Day.\n" );
      if ( time.t_day==14 && time.t_month==6 )
         printf("It's Flag Day.\n");
      if ( time.t_day==21 && time.t_month==6 )
         printf("It's the First Day of 
Summer.\n");
      if ( time.t_day==4 && time.t_month==7 )
         printf("Happy 4th of July!\n");
      if ( time.t_day==22 && time.t_month==9 )
         printf("It's the First Day of 
Autumn.\n");
      if ( time.t_day==31 && time.t_month==10 )
         printf("Trick or Treat, it's 
Halloween!\n");
      if ( time.t_day==21 && time.t_month==12 )
         printf("It's the First Day of 
Winter.\n");
      if ( time.t_day==25 && time.t_month==12 )
         printf("Merry Christmas!\n");
   }
}


/* Help for date */
dhelp()
{
   printf("Usage: \n");
   printf("    date [-t] [-m] [-d] [-g] [-h]\n");
   printf("        -t = with the time\n");
   printf("        -m = time in military 
notation\n");
```

```
    printf("       -d = with the day of the week\n");
    printf("       -g = with a greeting\n");
    printf("       -h = with holiday greeting\n");
    exit( 0 );
}


/* Print the day of the week */
pday( t )
struct sgtbuf *t;
{

/* Variables used */
    int n;
    register int i;
    int fudge=3; /* My fudge factor */

    static char *day[] = {
       "Sunday",
       "Monday",
       "Tuesday",
       "Wednesday",
       "Thursday",
       "Friday",
       "Saturday"
    };

    static int day_count[] = {
       31, 28, 31, 30,
       31, 30, 31, 31,
       30, 31, 30, 31
    };

/* Calculate today's day of the week */
    n = (*t).t_year + fudge + ((*t).t_year+3)/4;
    for ( i=0; i<(*t).t_month ; i++ )
       n+=day_count[i];
    n+=(*t).t_day;

/* Adjust if this is leap year */
    if ( ((*t).t_year % 4 )==0 && (*t).t_month>2 )
          n++;

/* Print day of the week */
    printf("%s  ", day[ n % 7 ]);

}

/* Print the date */
pdate( t )
struct sgtbuf *t;
{

/* The 12 months */
    static char *month[] = {
       "Unknown",
       "Jaunary",
       "February",
       "March",
       "April",
       "May",
       "June",
       "July",
       "August",
       "September",
       "October",
       "November",
       "December"
    };

    printf("%s %2d, 19%02d\n", month[(*t).t_month],
             (*t).t_day, (*t).t_year );

}

/* Print the time */
ptime( t, pf )
struct sgtbuf *t;
int pf;
{

    if ( (*t).t_hour<12 )
       pf=FALSE;
    if ( (*t).t_hour>12 )
       (*t).t_hour-=12;
    printf("%2d:%02d:%02d ", (*t).t_hour,
            (*t).t_minute, (*t).t_second );
    if ( pf )
       printf("PM\n");
    else
       printf("AM\n");
}

/* Print in military time */
pmtime( t )
struct sgtbuf *t;
{
    printf("%02d:%02d:%02d\n", (*t).t_hour,
            (*t).t_minute, (*t).t_second );
}

```

+++

SOFTWARE USER NOTES

A Tutorial Series

By : Ronald W Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

From Basic Assembler to HLL's

OS9 - SK*DOS Text File Interchange

I have been running both SK*DOS and OS9 on the 68008 machine for some time, and for various reasons I wanted to be able to move text files back and forth between the two systems. SK*DOS being considerably simpler than OS9, I decided I would work at the project from the SK*DOS end and leave the OS9 end alone. Therefore it was necessary to work with an OS9 formatted disk. The first problem is an incompatibility with the way side select works between OS9 and SK*DOS. It was immediately necessary to limit the disk to single sided, though 80 tracks and double density work nicely. I decided to go for double density in order to allow the most text on a disk. Since the programs are based on a number of assumptions including the number of sectors per track, the double density becomes a necessity for a disk to work with these programs.

I got out my Sherlock Holmes hat and sat down to see what I could find out. (Holmes would have called this a "ten pipe" problem but since I don't smoke it became a two liter Cherry Coke and bag of pretzels problem). The easy half of the project is to write a file with OS9 and figure out how to read it into an SK*DOS file. I decided to tackle that first. The most valuable tool I have for this project is the SDUMP program that I wrote first. SDUMP is a take-off on the MDUMP and DDUMP programs published here last time. SDUMP, however works strictly with reading sectors from the disk. The user specifies the track and sector and it is read and displayed. With this utility, I was able to examine the first sectors on track zero of the OS9 disk and find the directory sectors. I found that there are 8 entries per sector, except the first which has six. A good start at the project would be a directory utility that would read the filenames and starting sectors for each file from the OS9 disk. What follows are the somewhat incoherent notes that I took in the process of getting the job done:

OS9 Disk Format

This is an attempt to analyze the disk format of an OS9 formatted disk to see if it is feasible to copy ASCII files from OS9 to SK*DOS. Because of differences in side selection, the OS9 disk is formatted single sided. In order to limit the problem a little, I chose double density and 80 tracks though 40 would work fine. I found that in order to format a disk single sided I just had to answer "N" to the prompt in OS9 FORMAT that asks if it is OK to proceed. OS9 then prompts for single or double density, single or double sided, number of tracks, and ask if format should be changed to 48 TPI (in simple terms that means "do you want double stepping?).

Once the disk was formatted I copied some files to it. (Still in OS9 of course). For testing purposes I copied my source CJUST.C, STDIO.H, and CTYPE.H files to / D0 root directory.

Dumping track 00 sector 03 showed a directory. See dump of 0003. The first $40 bytes (00 to 3F) contained some information that I didn't need to interpret. The directory starts at byte $40 and every 32 bytes thereafter. If enough files are written, it will continue in the next sector, 00 04 etc.

A little investigation revealed that bytes 30 and 31 ($1E and 1F) of the directory entry are the 16 bit sector number of the first sector of the file. The sector number is not quite the same as the track and sector number. OS9 formats track zero with 10 sectors. 0000 through 0009. The remaining tracks have 16 sectors each. Unfortunately sector count 0A refers to track 01 sector 00 which is the next sector after track/sector 0009. If we add 6 to the sector count, the last hex digit is the sector number and what precedes it is the track. For example, the directory on t/s 0003 shows that CJUST.C file starts at sector 000B. Add 6 and we get 0011. The 001 means track 01 and the last digit 1 means sector 01.

I dumped track/ sector 0101 we get the information sector for that file. (See dump for 0101). I don't know what the folks at Microware call this sector for a sequential file, but I'll refer to it as the File Information Record or FIR for short.

Byte 00 of the FIR is the attribute byte $3F in the present case meaning directory. That is, if an attribute is on, the bit position in that byte is a 1. If not, it is a zero. Byte 01 and 02 are zeros in all the records that I have looked at. Possibly they have to do with sub directories. Bytes 03 - 05 are the year, month, and day in that order. $57 translates to decimal 87. $0C is 12 and $01 is 1 so the date is 87/12/1 or December 1 1987. The next two bytes (06 and 07) are the hour and minute again in Hex. $0A is 10 and $08 is of course 8 so the time is 10:08. This is 24 hour time, I found by examining the information sector for another file where the hour is $11 and the minute is $2A, which translate to 17:42 or 5:42 PM.

Bytes 08 and 09 code something we don't need, perhaps the owner's code. Bytes $0A and $0B contain $0035 which is 53 decimal. There are 53 complete sectors in the text that follows. The 54th is the last and partial sector. Byte $0C contains the count of characters in the last sector! the $0F indicates 15 characters. A check of the last sector of this file 0500 shows that bytes 00 to 0E inclusive are data. That is a count of $0F or 15.

The last three bytes of the first group of 16 seem to be a repeat of the date information we found in bytes $03 to $06. Perhaps one is the creation date and the other the date of the most recent update. That doesn't matter much to the object of this exercise. Bytes $12 and $13 contain the sector number ($0C) of the first data sector. In this case, we add 6 and get $0012 so we know that if we dump track 01 sector 02 we will see the first sector full of data. The last item in the information sector is the total number of sectors including the last partial one. Bytes $13 and $14 contain that count. (I am assuming that files over 255 sectors long have the count overflow into the next higher byte $13.) In the present case the HEX 36 translates to decimal 54 sectors. That agrees with the earlier 53 sector plus $0F byte count. Actually that is really $00350F bytes in the first line, since in this case the sectors contain 256 bytes, the interpretation is arbitrary.

After figuring that out, I knew how to read a filename in the directory and find out what track and sector containwd the information record. I knew how to read the information record to find out the sector count and number of bytes to read in the last sector, so I was all set to dump a sector and see what was there. It turned out that the SK*DOS SREAD function did just fine. All I had to do is point A4 at an FCB, put the drive number in byte 3 of it, the track in byte 34 and the sector in byte 35. Now I called SREAD and the result was what is shown in the dumped data sector. Note first of all that unlike FLEX or SK*DOS, there are no sector linkage bytes in the sector. There are 256 bytes of data. At first glance that looks very efficient, but remember that a whole sector was used for the FIR. I strongly suspect that as files get fractured due to repeated file deletions and additions, a whole lot more of that sector is used up as a sector map. I'll stick to freshly formatted disks and files copied with no deletions for this exercise, however. I'll also not bother investigating the complexities of random files.

About then I figured it was about time to start writing an SK*DOS utility to do a directory and add the $06 to the sector count to display the starting track and sector for each file. Then I figured I could write a simple copy utility that could read the information sector and read the OS9 file, writing it to an SK*DOS file. Assuming that all directories for this disk configuration start at t/s 0003 and are sequential. I'd read the directory until the first byte of what should be the next entry is a null. Probably, when the directory overflows, OS9 allocates some sectors elsewhere and puts them in a sector map that we could follow, but I decided that since the purpose of the disk in question is to transfer some files from OS9 to SK*DOS, I would be careful not to overflow the normal directory sectors and thus simplify the problem.

There is room for 6 directory entries on sector 0003 and for 8 entries each on sectors 4 through 9 for a total of 54 files before the sequential directory sectors on track 0 get full. That should be more than adequate for transferring files from OS9 to SK*DOS.

Listing 1 shows the directory utility which reports the filename and the starting sector (the FIR) for each file in the directory. The utility is called OS9DIR. The next step was to write an OS9COPY utility that is given the track and sector (as presented in the OS9DIR utility) and a filename to which to write the text on the SK*DOS disk.

OS9COPY is the result of an evening's programming and debug. Using the procedure outlined above, it copies files from an OS9 formatted disk and writes them to a standard SK*DOS file. OS9 normally does not use space compression. That is, if the text file contains 20 spaces there are 20 space characters in the file. However, I did find one file with horizontal tab characters $09 in it, and it did not copy correctly to the SK*DOS disk. The file was part of A "C" compiler (the STDIO.H file). Perhaps the compiler uses compressed files. Any file that I had edited under OS9 copied over with no problems. Perhaps someday I'll combine OS9DIR and OS9COPY and add a string match routine so it can copy a named file from an OS9 disk to a named file on an SK*DOS disk. Meanwhile, the two utilities together are very handy.

Sector dumps follow:

```
Track 00 Sector 03
(first directory
sector)

00  2E AE 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02
20  AE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02
40  63 6A 75 73 74 2E E3 00 00 00 00 00 00 00 00 00  cjust.c
50  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0B
60  73 74 64 69 6F 2E E8 00 00 00 00 00 00 00 00 00  stdio.h
70  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 42
80  63 74 79 70 65 2E E8 00 00 00 00 00 00 00 00 00  ctypc.h
90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4B
A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


Track 01 Sector 01     serial sector 0B FIR of cjust.c

00  3F 00 00 57 0C 01 0A 08 01 00 00 35 0F 57 0C 01
10  00 00 0C 00 36 00 00 00 00 00 00 00 00 00 00 00

Remainder of sector is all 00

Track 01 Sector 02   first data sector of cjust.c

00  2F 2A 20 4A 55 53 54 20 74 65 78 74 20 66 6F 72
/* JUST text for
10  6D 61 74 74 65 72 20 66 6F 72 20 43 65 6E 74 72
matter for Centr
```

etc. Data runs to last byte of sector and multiple spaces are not compressed in any way.

Having finished the above utilities, I decided to try the more ambitious project of writing a utility to copy files from SK*DOS to OS9 on the OS9 formatted disk. That was a little more challenge. I started out the hard way to find the last used sector. I found the last directory entry, went to the FIR for that file, added the sectors used to the starting sector to arrive at the next unused sector. I noted that last directory entry position and figured I would write the filename and starting sector there after I had written the file.

Later, when I had written a file to the OS9 disk with SK*DOS (successfully), I found that OS9 didn't even see the directory entry. A little sector dumping found that track 00 sector 02 has a pointer to the next empty directory entry position. When the disk was first formatted a dump of that sector showed bytes 0B and 0C of the sector to contain $0040. Byte 40 of the first directory sector (03) is the location for the first directory entry. When I copied a file to the disk from OS9 those bytes changed to $0060 and the directory entry appeared at location $40 in sector 3 of track 0. I booted up OS9 and wrote a number of files to the disk, each time watching that value increment by $20. It reached 00E0 and the next file that I wrote made it increase to 0100. It was clear that the value in byte $0B, when 3 is added to it, is the sector number and the next byte is the pointer to the address in the sector for the next directory entry. I decided that I could rewrite a little to pick up the first empty directory location by reading these bytes, though I delayed that for finishing the initial version. I fixed my SK*DOS write program to bump this pointer by $20 and sure enough, OS9 then listed the directory entry, but it couldn't open the file.

I looked more closely and found that the OS9 written directory entries had the high order bit of the last character set. Some software I've seen uses this technique to signal the end of a string without wasting an extra character as a terminator. I fixed my program to do that and wrote another file to the disk while running SK*DOS. OS9 would now recognize the file and could open and read it.

Now I discovered another feature of OS9. When I wrote another file to the disk from OS9 it overwrote the file I had just written to it in SK*DOS. A dump of track 00 sector 01 showed a pattern of bits that changed when I wrote a file to the disk from OS9. I puzzled over it for some time until I realized that there are 160 bytes in the map and that each byte is 8 bits. It became apparent that the bits in this sector correlate 1 to 1 with which sectors are in use and which are free. A 1 in a bit indicates the corresponding sector is not available for user files. The first two bytes in the map for a freshly formatted disk are $FFE0. Each $F is 4 1's so each byte containing $FF indicates 8 sectors in use. The $E is binary 1110 for three more sectors. There are 11 ones. Decimal 11 is $0B, the first available sector as we determined previously, the sector where the first FIR goes. I guess we could think of the 11 ones as representing serial sectors 00 through $0A meaning that the next available sector is $0B. I wrote some more files to the disk from OS9 and verified that another 1 appeared in the map for each sector used completely or partially including the FIR sector.

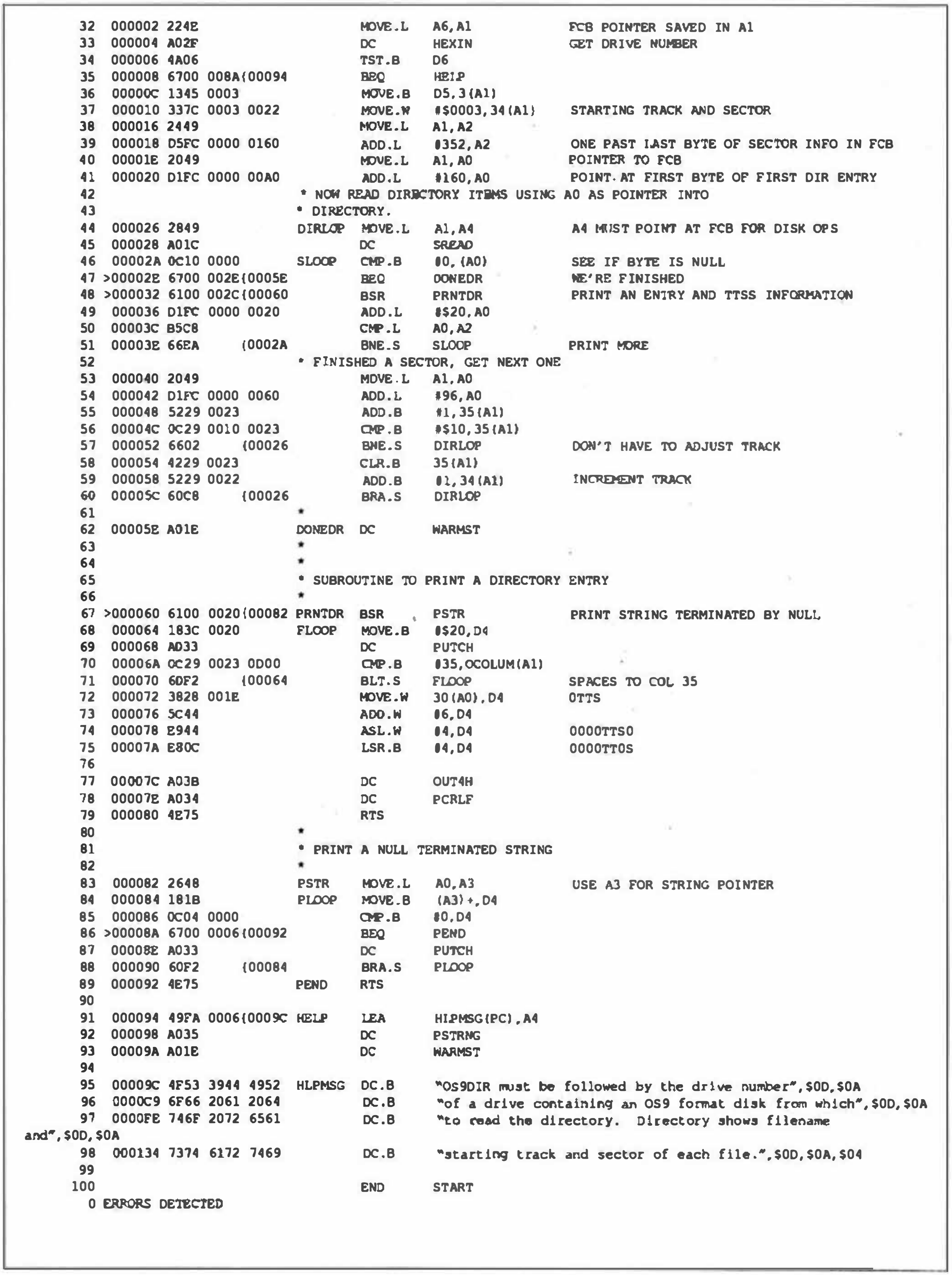
I added the appropriate code to write 1's to the sector map in the SK*DOS program, forgetting the extra 1 for the FIR. I wrote a file to the disk and I could read it using OS9. I wrote a file from OS9 and the previous file lost its end. When I fixed my sector count for the FIR sector the problem was cured and I had the whole utility running. There was still a small problem. Since the program read the directory to find the last FIR and then adds sectors to find the first available one, SK*DOS didn't know what to do with a freshly formatted and blank disk. There were no directory entries, so it was lost. A couple of days later, I fixed the program to count 1's in the sector map to find the first sector and to use the pointer to find the place for the next directory entry. Needless to say, it worked.

Listings for the three utilities are included with this. requires one argument, the number of the drive containing the OS9 disk. I was tempted to default to the working drive but that doesn't work well with hard disk systems, so I chose to let the user supply the drive number.

OS9COPY requires first the drive number, then the track and sector of the start of the file (as displayed by the OS9DIR utility), and lastly the name of the SK*DOS file. In this case working drive and .TXT are defaults.

COS9 requires the SK*DOS filename (default .TXT and working drive) the drive number for the OS9 disk, and the OS9 filename. Files from SK*DOS written to the disk do not make it incompatible with OS9 in any way. Files may be transferred back and forth until the "transfer disk" is full, at which point it must be reformatted.

```
                        ************************************************************
                        * OS9 DISK DIRECTORY PROGRAM                               *
                        *                                                          *
                        * FORMAT: OS9DIR N  WHERE N IS THE DRIVE NUMBER            *
                        * OF THE DRIVE CONTAINING A SINGLE SIDED OS9 FORMATTED DISK *
                        *                                                          *
                        * OUTPUT WILL SHOW THE TRACK AND SECTOR FOR THE INFORMATION *
                        * RECORD OF EACH FILE.. THIS INFORMATION TO BE USED BY     *
                        * AN OS9 TO SK*DOS ASCII FILE COPY PROGRAM                 *
                        ************************************************************

                        * SK*DOS / 68K EQUATES FOR USER PROGRAMS

            00000D00    OCOLUM  EQU     3328          Offset from start of FCB
            0000A02F    HEXIN   EQU     $A02F         Input hex number from command line
            0000A029    GETCH   EQU     $A029         Get input character with echo (7 bits)
            0000A01C    SREAD   EQU     $A01C         Read a track and sector
            0000A03B    OUT4H   EQU     $A03B         Output 4 hex digits
            0000A034    PCRLF   EQU     $A034         Print CR/LF
            0000A036    PNSTRN  EQU     $A036         Print string (Without CR/LF)
            0000A035    PSTRNG  EQU     $A035         Print CR/LF and string
            0000A033    PUTCH   EQU     $A033         Output character
            0000AD00    VPOINT  EQU     $A000         Point to SK*DOS variable area
            0000A01E    WARMST  EQU     $A01E         Warm start
                        *
                        *
000000 A000             START   DC      VPOINT        GET POINTER
```



```
 000002 224E                     MOVE.L   A6,A1            FCB POINTER SAVED IN A1
 000004 A02F                     DC       HEXIN            GET DRIVE NUMBER
 000006 4A06                     TST.B    D6
 000008 6700 008A(00094          BEQ      HELP
 00000C 1345 0003                MOVE.B   D5,3(A1)
 000010 337C 0003 0022           MOVE.W   #$0003,34(A1)    STARTING TRACK AND SECTOR
 000016 2449                     MOVE.L   A1,A2
 000018 D5FC 0000 0160           ADD.L    #352,A2          ONE PAST LAST BYTE OF SECTOR INFO IN FCB
 00001E 2049                     MOVE.L   A1,A0            POINTER TO FCB
 000020 D1FC 0000 00A0           ADD.L    #160,A0          POINT AT FIRST BYTE OF FIRST DIR ENTRY
                        * NOW READ DIRECTORY ITEMS USING A0 AS POINTER INTO
                        * DIRECTORY.
 000026 2849            DIRLOP   MOVE.L   A1,A4            A4 MUST POINT AT FCB FOR DISK OPS
 000028 A01C                     DC       SREAD
 00002A 0C10 0000       SLOOP    CMP.B    #0,(A0)          SEE IF BYTE IS NULL
>00002E 6700 002E(0005E          BEQ      DONEDR           WE'RE FINISHED
>000032 6100 002C(00060          BSR      PRNTDR           PRINT AN ENTRY AND TTSS INFORMATION
 000036 D1FC 0000 0020           ADD.L    #$20,A0
 00003C B5C8                     CMP.L    A0,A2
 00003E 66EA      (0002A         BNE.S    SLOOP            PRINT MORE
                        * FINISHED A SECTOR, GET NEXT ONE
 000040 2049                     MOVE.L   A1,A0
 000042 D1FC 0000 0060           ADD.L    #96,A0
 000048 5229 0023                ADD.B    #1,35(A1)
 00004C 0C29 0010 0023           CMP.B    #$10,35(A1)
 000052 6602      (00026         BNE.S    DIRLOP           DON'T HAVE TO ADJUST TRACK
 000054 4229 0023                CLR.B    35(A1)
 000058 5229 0022                ADD.B    #1,34(A1)        INCREMENT TRACK
 00005C 60C8      (00026         BRA.S    DIRLOP
                        *
 00005E A01E            DONEDR   DC       WARMST
                        *
                        *
                        * SUBROUTINE TO PRINT A DIRECTORY ENTRY
                        *
>000060 6100 0020(00082 PRNTDR   BSR      PSTR             PRINT STRING TERMINATED BY NULL
 000064 183C 0020       FLOOP    MOVE.B   #$20,D4
 000068 A033                     DC       PUTCH
 00006A 0C29 0023 0D00           CMP.B    #35,OCOLUM(A1)
 000070 6DF2      (00064         BLT.S    FLOOP            SPACES TO COL 35
 000072 3828 001E                MOVE.W   30(A0),D4        0TTS
 000076 5C44                     ADD.W    #6,D4
 000078 E944                     ASL.W    #4,D4            0000TTS0
 00007A E80C                     LSR.B    #4,D4            0000TT0S

 00007C A03B                     DC       OUT4H
 00007E A034                     DC       PCRLF
 000080 4E75                     RTS
                        *
                        * PRINT A NULL TERMINATED STRING
                        *
 000082 2648            PSTR     MOVE.L   A0,A3            USE A3 FOR STRING POINTER
 000084 181B            PLOOP    MOVE.B   (A3)+,D4
 000086 0C04 0000                CMP.B    #0,D4
>00008A 6700 0006(00092          BEQ      PEND
 00008E A033                     DC       PUTCH
 000090 60F2      (00084         BRA.S    PLOOP
 000092 4E75            PEND     RTS

 000094 49FA 0006(0009C HELP     LEA      HLPMSG(PC),A4
 000098 A035                     DC       PSTRNG
 00009A A01E                     DC       WARMST

 00009C 4F53 3944 4952  HLPMSG   DC.B     "OS9DIR must be followed by the drive number",$0D,$0A
 0000C9 6F66 2061 2064           DC.B     "of a drive containing an OS9 format disk from which",$0D,$0A
 0000FE 746F 2072 6561           DC.B     "to read the directory.  Directory shows filename
and",$0D,$0A
 000134 7374 6172 7469           DC.B     "starting track and sector of each file.",$0D,$0A,$04


                                 END      START
  0 ERRORS DETECTED
```

```


                                  *****************************************************
                                  * OS9 FIIE COPY PROGRAM                             *
                                  *                                                   *
                                  * FORMAT: OS9COPY N TTSS FILENAME                   *
                                  * THIS UTILITY WILL COPY A FILE FROM AN OS9 FORMATTED DISK  *
                                  * (SINGLE SIDED).  THE TTSS ARE TRACK AND SECTOR OF THE FILE *
                                  * INFORMATION RECORD AS DETERMINED BY THE OS9DIR UTILITY.   *
                                  * N IS THE NUMBER OF THE DRIVE CONTAINING THE OS9 DISK.     *
                                  *                                                   *
                                  * FILE WILL BE COPIED TO THE SK*DOS WORKING DRIVE.  *
                                  * THE DEFAULT EXTENSION IS .TXT                     *
                                  *****************************************************

                                  * SK*DOS / 68K EQUATES FOR USER PROGRAMS

                00000D00          OCOLUM   EQU      3328                Offset from start of FCB
                0000A02F          HEXIN    EQU      $A02F               Input hex number from command line
                0000A023          GETNAM   EQU      $A023               Get file name from command line
                0000A024          DEFEXT   EQU      $A024               Default extension
                0000A006          FOPENW   EQU      $A006               Open file for write
                0000A002          FWRITE   EQU      $A002               Write byte to file
                0000A008          FCLOSE   EQU      $A008               Close file
                0000A029          GETCH    EQU      $A029               Get input character with echo (7 bits)
                0000A01C          SREAD    EQU      $A01C               Read a track and sector
                0000A03B          OUT4H    EQU      $A03B               Output 4 hex digits
                0000A034          PCRIF    EQU      $A034               Print CR/IF
                0000A036          PNSTRN   EQU      $A036               Print string (without CR/IF)
                0000A035          PSTRNG   EQU      $A035               Print CR/IF and string
                0000A033          PUTCH    EQU      $A033               Output character
                0000A000          VPOINT   EQU      $A000               Point to SK*DOS variable area
                0000A01E          WARMST   EQU      $A01E               Warm start
                                  *
000000                                     ORG      0
000000                            OUTFCB   DS.B     608
000260                            OSECTS   DS.W     1
000262                            OBYTES   OS.B     1
                00000263          LAST     EQU      *
000000                                     ORG      0
                                  *
                                  * GET STUFF FROM COMMAN0 LINE REGARDING INPUT FIIE
                                  *
000000 A000                       START    DC       VPOINT              GET POINTER
000002 224E                                MOVE.L   A6,A1               FCB POINTER SAVED IN A1
000004 A02F                                DC       HEXIN               GET DRIVE NUMBER
000006 4A06                                TST.B    06
000008 6700 02F4{002FE                     BEQ      HEIP
00000C 1345 0003                           MOVE.B   05,3(A1)
000010 A02F                                DC       HEXIN               GET TRACK AND SECTOR OF FIR
000012 3345 0022                           MOVE.W   05,34(A1)           STARTING TRACK AND SECTOR
000016 2849                                MOVE.L   A1,A4               A4 MUST POINT AT FCB FOR DISK OPS
000018 A01C                                DC       SREAD
                                  *
                                  * NOW GET OUTPUT FIIE READY
                                  *
00001A 47FA 0068{00084                     LEA      VARS(PC),A3         POINT A3 AT VARIABLE AREA
00001E 49EB 0000                           LEA      OUTFCB(A3),A4       POINT A4 AT OUTFCB
000022 A023                                DC       GETNAM              GET OUTPUT FILE NAME
000024 183C 0001                           MOVE.B   #1,D4               DEFAULT TO .TXT EXTENSION
000028 A024                                DC       DEFEXT
00002A A006                                DC       FOPENW
                                  *
                                  * NOW GET INPUT FILE SECTOR AND BYTE COUNT
                                  *
00002C 3769 006A 0260                      MOVE.W   106(A1),OSECTS(A3) SECTOR COUNT IN MEMORY
000032 1769 006C 0262                      MOVE.B   108(A1),OBYTES(A3) BYTE COUNT FOR LAST SECTOR
000038 6100 02AE{002E8 READIP   BSR      NEXSEC              NEXT SECTOR
00003C 303C 0100                           MOVE.W   #256,D0             COUNTER FOR ONE SECTOR
000040 2849                                MOVE.L   A1,A4
000042 2049                                MOVE.L   A1,A0
000044 D1FC 0000 0060                      ADD.L    #96,A0              POINT AT FIRST BYTE
00004A A01C                                DC       SREAD               READ A SECTOR
00004C 49EB 0000                           LEA      OUTFCB(A3),A4       POINT AT OUTFCB
```

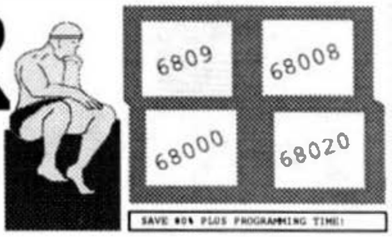
```
000050 1818            READ02   MOVE.B   (A0)+,D4
000052 A002                     DC       FWRITE           WRITE BYTE TO OUTPUT FILE
000054 5340                     SUB.W    #1,D0            BYTE COUNT
000056 66F8      {00050         BNE.S    READ2
000058 536B 0260                SUB.W    #1,OSECTS(A3)
00005C 66DA      {00038         BNE      READLP
                       * DONE READING WHOLE SECTORS
00005E 6100 0288{002E8          BSR      NEXSEC           GO TO LAST SECTOR
000062 4240                     CLR.W    D0
000064 102B 0262                MOVE.B   OBYTES(A3),D0
000068 2849                     MOVE.L   A1,A4
00006A 2049                     MOVE.L   A1,A0
00006C D1FC 0000 0060           ADO.L    #96,A0
000072 A01C                     DC       SREAD
000074 49EB 0000                LEA      OUTFCB(A3),A4    SET A4 TO POINT AT FCB
000078 1818            READ3    MOVE.B   (A0)+,D4
00007A A002                     DC       FWRITE
00007C 5340                     SUB.W    #1,D0
00007E 66F8      {00078         BNE.S    READ3
000080 A008                     DC       FCLOSE           CLOSE OUTPUT FILE
000082 A01E                     DC       WARMST
                       * VARIABLE AREA
000084                 VARS     DS.W     0                DUMMY TO CAUSE WORD ALIGNMENT
000084                          DS.B     LAST             RESERVE BYTES FOR VARIABLES HERE
                       *
                       * SUBROUTINE NEXSEC NEXT SECTOR
                       *
0002E8 5229 0023       NEXSEC   ADD.B    #1,35(A1)
0002EC 0C29 0010 0023           CMP.B    #$10,35(A1)      END OF TRACK
0002F2 6608      {002FC         BNE.S    DONSEC
0002F4 4229 0023                CLR.B    35(A1)           SECTOR ZERO
0002F8 5229 0022                ADD.B    #1,34(A1)        NEXT TRACK
0002FC 4E75            DONSEC   RTS

0002FE 49FA 0006{00306 HELP     LEA      HLPMSG(PC),A4
000302 A035                     DC       PSTRNG
000304 A01E                     DC       WARMST

000306 466F 726D 6174  HLPMSG   DC.B     "Format: OS9COPY 1 0101 SKFILENAME",$0D,$0A
000329 4F53 3943 4F50           DC.B     "OS9COPY copies a text file from an OS9 formatted",$0D,$0A
00035B 6469 736B 2E20           DC.B     "disk.  The single digit following OS9COPY is the",$0D,$0A
00038D 6C6F 6769 6361           DC.B     "logical drive number for the drive containing the",$0D,$0A
0003C0 4F53 3920 6469           DC.B     "OS9 disk.  The four digit number is the track and",$0D,$0A
0003F3 7365 6374 6F72           DC.B     "sector number of the OS9 file as determined by the",$0D,$0A
000427 4F53 3944 4952           DC.B     "OS9DIR utility.  The SKFILENAME defaults to the",$0D,$0A
000458 776F 726B 696E           DC.B     "working drive and .TXT extension.",$0D,$0A,$04

                                END      START
0 ERRORS DETECTED



                       ****************************************************************
                       * COPY SK*DOS FILE TO OS9 DISK PROGRAM                         *
                       *                                                              *
                       * FORMAT: COS9 SKFILENAME DR OSFILENAME  WHERE OR IS #         *
                       * OF THE DRIVE CONTAINING A SINGLE SIDED OS9 FORMATTED DISK    *
                       *                                                              *
                       * THIS PROGRAM WILL WRITE THE FILE TO THE DISK AND MAKE A      *
                       * FIR SECTOR AND DIRECTORY ENTRY FOR ONE FILE AT A TIME        *
                       ****************************************************************

                       * SK*DOS / 68K EQUATES FOR USER PROGRAMS
                       *
            00000D00   OCOLUM   EQU      3328             Offset from start of FCB
            000002EE   CMONTH   EQU      750              Month byte binary
            000002EF   CDAY     EQU      751              Day byte binary
            000002F0   CYEAR    EQU      752              Year byte binary
            0000A03F   GETDNT   EQU      $A03F            Get date and time
            0000A02F   HEXIN    EQU      $A02F            Input hex number from command line
            0000A029   GETCH    EQU      $A029            Get input character with echo (7 bits)
            0000A023   GETNAM   EQU      $A023            Get filename from command lint to FCB
            0000A02D   GETNXT   EQU      $A02D            Get next char of command line
```

```
          0000A024     OEFEXT  EQU      $A024              Default extension
          0000A005     FOPENR  EQU      $AD05              Open file for read
          0000A008     FCLOSE  EQU      $A008              Close file, A4 points at FCB
          0000A001     FREAD   EQU      $A001              Read char from file open for read
          0000A002     FWRITE  EQU      $A002              Write char to file open for write
          0000A037     PERROR  EQU      $A037              Print file error message
          0000A01C     SREAD   EQU      $A01C              Read a sector
          0000A01D     SWRITE  EQU      $A01D              Write a sector
          0000A03B     OUT4H   EQU      $A03B              Output 4 hex digits
          0000A034     PCRLF   EQU      $A034              Print CR/LF
          0000A036     PNSTRN  EQU      $A036              Print string (Without CR/LF)
          0000A035     PSTRNG  EQU      $A035              Print CR/LF and string
          0000A033     PUTCH   EQU      $A033              Output character
          0000A000     VPOINT  EQU      $A000              Point to SK*DOS variable area
          0000A01E     WARMST  EQU      $A01E              Warm start
                       *
                       *
                       * GET SK*OOS FILENAME FROM COMMAND LINE AND OPEN FOR READ
                       *
000000 49FA 033E{00340 START   LEA      INFCB(PC),A4
000004 A023                    DC       GETNAM
000006 6500 021E{00226         BCS      HELP
00000A 183C 0001               MOVE.B   #1,D4
00000E A024                    DC       DEFEXT
000010 6500 0210(00222         BCS      ERROR
000014 A005                    DC       FOPENR
000016 6500 020A(00222         BCS      ERROR
                       *
                       * NOW GET OS9 DRIVE NUMBER
                       * LEAVE OS9 FIIENAME FOR LATER
                       *
00001A A000                    DC       VPOINT             GET POINTER
00001C 284E                    MOVE.L   A6,A4              FCB POINTER
00001E A02F                    DC       HEXIN              GET DRIVE NUMBER
000020 1945 0003               MOVE.B   D5,3(A4)
                       * SET UP BUFFER POINTER LIMIT (END OF SECTOR BUFFER)
000024 244C                    MOVE.L   A4,A2              FIRST SET UP SECTOR BUFFER LIMIT
000026 D5FC 0000 0160          ADO.L    #352,A2            ONE PAST LAST BYTE OF SECTOR INFO IN FCB
                       *
                       * GET POINTER FOR NEXT DIRECTORY ENTRY
                       * FROM BYTES 11 AND 12 OF SECTOR 0002 AND SAVE IT IN DIRTS
                       *
00002C 204C                    MOVE.L   A4,A0              POINTER TO FCB
00002E 01FC 0000 0060          ADO.L    #96,A0             POINT AT FIRST BYTE OF FIRST OIR ENTRY
000034 397C 0002 0022          MOVE.W   #$0002,34(A4)      POINT AT SECTOR CONTAINING OIR PTR
00003A A01C                    DC       SREAD              READ THE INFO SECTOR
00003C 4280                    CIR.L    00
00003E 1028 000B               MOVE.B   11(A0),D0          WORD MISALIGNED IN BUFFER
000042 E140                    ASL.W    #8,D0
000044 1028 000C               MOVE.B   12(A0),D0          GET LO ORDER BYTE OF WORD
000048 0640 0300               AOD.W    #$0300,D0          DIR STARTS AT SECTOR 3 OSBB
00004C 47FA 02EE{0033C         IEA      DIRTS(PC),A3
000050 3680                    MOVE.W   D0,(A3)            SAVE IN DIRTS
                       *
                       * FIND NEXT AVAIIABLE SECTOR FROM SECTOR BIT MAP
                       * IN 0001 BY COUNTING BITS
                       *
000052 4280                    CIR.L    D0
000054 4281                    CIR.L    D1
000056 4282                    CIR.L    D2
000058 397C 0001 0022          MOVE.W   #$0001,34(A4)
00005E A01C                    DC       SREAD
000060 2018                    MOVE.L   (A0)+,D0
000062 343C 0020       FN00    MOVE.W   #32,D2
000066 E380            FN01    ASL.L    #1,D0
000068 640A      {00074        BCC.S    DONFND
00006A 5241                    ADO.W    #1,01              SECTOR COUNT
00006C 5342                    SUB      #1,D2              COUNTER FOR LONG WORD
00006E 66F6      {00066        BNE      FND1
000070 2018                    MOVE.L   (A0)+,D0           GET FOUR MORE BYTES OF SECTOR MAP
000072 60EE      {00062        BRA      FND0               RELOAD LONG WORD COUNT
000074 47FA 02C2(00338 DONFND  LEA      FIRSEC(PC),A3
000078 5C41                    ADD.W    #6,D1              SECTOR COUNT TO TRACK SECTOR CONVERT OTTS
00007A E941                    ASL.W    #4,D1              TTSO
```

```
00007C E809                         LSR.B   #4,D1              TTOS
00007E 3681                         MOVE.W  D1,(A3)            SAVE FOR LATER
                        *
                        * GET READY TO WRITE FIRST SECTOR OF DATA
                        *
000080 3941 0022                    MOVE.W  D1,34(A4)          PUT IN FCB
000084 61D0 0186(0020C              BSR     NEWTS              GET NEXT SECTOR AS FIRST WRITE SECTOR
000088 204C                         MOVE.L  A4,A0
00008A D1FC 0000 0160               ADD.L   #352,A0            POINT A0 AT SECTOR DATA (0+256)
000090 4280                         CLR.L   D0                 SET UP AS BYTE COUNTER
000092 91FC 0000 0100   DOSECT      SUB.L   #256,A0            RESET TO START OF BUFFER
000098 49FA 02A6(00340              LEA     INFCB(PC),A4       POINT A4 AT SK INPUT FILE FCB
                        *
                        * WRITE A SECTOR LOOP FOLLOWS
                        *
00009C A001             SECT        DC      FREAD              READ A CHAR
00009E 6614      (000B4             BNE.S   DONWRT             ASSUME ERROR IS EOF
0000A0 5280                         ADD.L   #1,D0              BYTE COUNTER
0000A2 10C5                         MOVE.B  D5,(A0)+           PUT IN OUTPUT SECTOR BUFFER
0000A4 B5C8                         CMP.L   A0,A2
0000A6 66F4      (0009C             BNE.S   SECT
0000A8 A000                         DC      VPOINT
0000AA 284E                         MOVE.L  A6,A4              POINT AT OUTPUT BUFFER
0000AC A01D                         DC      SWRITE             WRITE THE SECTOR
0000AE 6100 015C(0020C              BSR     NEWTS              NEXT TRACK AND SECTOR
0000B2 60DE      (00092             BRA     DOSECT
                        *
                        * NOW HANDLE LAST PARTIAL SECTOR
                        *
0000B4 A000             DONWRT      DC      VPOINT
0000B6 284E                         MOVE.L  A6,A4
0000B8 10FC 00E5        DLOP        MOVE.B  #$E5,(A0)+
0000BC B5C8                         CMP.L   A0,A2
0000BE 66F8      (000B8             BNE.S   DLOP               FILL END OF BUFFER WITH $E5
0000C0 A01D                         DC      SWRITE
                        * NOW SET UP FIR DATA FOR WRITE
                        * FIR TRACK AND SECTOR IN FIRSEC
                        * LAST TRACK SECTOR WRITTEN IN 34(A4)
                        * BYTE COUNT (SAME AS SECTORS + BYTES) IN D0
                        * GET POINTER TO FCB AND SECTOR DATA (0)
0000C2 A000                         DC      VPOINT
0000C4 204E                         MOVE.L  A6,A0
0000C6 D0FC 0060                    ADD     #96,A0             POINT AT START OF SECTOR INFO
                        * NOW STUFF FIR WITH ALL THE NECESSARY DATA
0000CA 10FC 003F                    MOVE.B  #$3F,(A0)+         -EWREWR  00 PROTECTION CODES
0000CE 10FC 0000                    MOVE.B  #0,(A0)+           01
0000D2 10FC 0000                    MOVE.B  #0,(A0)+           02
0000D6 10EE 02F0                    MOVE.B  CYEAR(A6),(A0)+    03  DATE
00000A 10EE 02EE                    MOVE.B  CMONTH(A6),(A0)+   04   "
0000DE 10EE 02EF                    MOVE.B  CDAY(A6),(A0)+     05   "
0000E2 A03F                         DC      GETDNT             00HHMMSS IN D6 GET DATE AND TIME
0000E4 2206                         MOVE.L  D6,D1              KEEP IT SAFE FROM SK*DOS
0000E6 4841                         SWAP.W  D1                 MMSS00HH IN D1
0000E8 10C1                         MOVE.B  D1,(A0)+           06
0000EA E199                         ROL.L   #8,D1              SS00HHMM IN D1
0000EC 10C1                         MOVE.B  D1,(A0)+           07
0000EE 30FC 0100                    MOVE.W  #$0100,(A0)+       08-09
0000F2 E098                         ROR.L   #8,D0              BB000SSS IN D0
0000F4 30C0                         MOVE.W  D0,(A0)+           SECTORS 0A-0B
0000F6 E198                         ROL.L   #8,D0
0000F8 10C0                         MOVE.B  D0,(A0)+           BYTES IN PARTIAL SECTOR 0C
0000FA 10EE 02F0                    MOVE.B  CYEAR(A6),(A0)+    0D  DATE AGAIN
0000FE 10EE 02EE                    MOVE.B  CMONTH(A6),(A0)+   0E
000102 10EE 02EF                    MOVE.B  CDAY(A6),(A0)+     0F
000106 10FC 0000                    MOVE.B  #0,(A0)+           10
00010A 323A 022C(00338              MOVE.W  FIRSEC(PC),D1      STORED AS TTSS
00010E 3D41 0022                    MOVE.W  D1,34(A6)
000112 5B41                         SUB.W   #5,D1              CONVERT TO SECTOR NUMBER
000114 E909                         LSL.B   #4,D1              COMPRESS
000116 E849                         LSR.W   #4,D1              OPPOSITE OF TAKE APART
                        *
                        * FUSS BECAUSE WORD IS NOT WORD ALIGNED IN SECTOR BUFFER
                        *
000118 E099                         ROR.L   #8,D1              HI ORDER BYTE
```

```
00011A 10C1                      MOVE.B   D1,(A0)+         11
00011C E199                      ROL.L    #8,D1
00011E 10C1                      MOVE.B   D1,(A0)+         12
000120 E098                      ROR.L    #8,D0            SAME DEAL WITH TOTAL SECTORS
000122 5240                      ADD.W    #1,D0            ADD 1 FOR PARTIAL SECTOR
000124 43FA 0218(0033E           LEA      SCOUNT(PC),A1
000128 3280                      MOVE.W   D0,(A1)          SAVE TOTAL SECTOR COUNT
00012A E098                      ROR.L    #8,D0
00012C 10C0                      MOVE.B   D0,(A0)+         13
00012E E198                      ROL.L    #8,D0
000130 10C0                      MOVE.B   D0,(A0)+         14
000132 10FC 0000                 MOVE.B   #0,(A0)+         TO PUT IN WORD ALIGNMENT 15
000136 30FC 0000         FIRLOP  MOVE.W   #0,(A0)+         16-FF
00013A B5C8                      CMP.L    A0,A2
00013C 66F8      (00136          BNE.S    FIRLOP
00013E A01D                      DC       SWRITE           WRITE FIR SECTOR TO DISK
                         *
                         * WRITE THE DIRECTORY ENTRY FOR FILE
                         *
000140 303A 01FA(0033C           MOVE.W   DIRTS(PC),D0     SSBB PTR
000144 E048                      LSR.W    #8,D0            00SS
000146 3940 0022                 MOVE.W   D0,34(A4)
00014A 303A 01F0(0033C           MOVE.W   DIRTS(PC),D0     SSBB
00014E 0240 00FF                 AND.W    #$00FF,00        00BB
000152 204C                      MOVE.L   A4,A0
000154 D1FC 0000 0060            ADD.L    #96,A0           START OF SECTOR BUFFER
00015A D1C0                      ADD.L    D0,A0            NOW POINTS AT DIRECTORY ENTRY IN SEC. BUF
00015C A01C                      DC       SREAD            GET THE SECTOR
00015E 4280                      CLR.L    D0
000160 303A 01D6(00338           MOVE.W   FIRSEC(PC),D0
000164 E900                      ASL.B    #4,D0
000166 E848                      LSR.W    #4,D0            COMBINE TRACK AND SECTOR
000168 5D40                      SUB.W    #6,D0
00016A 3140 001E                 MOVE.W   D0,30(A0)        FIR SECTOR NUMBER
00016E A02D              NAMLOP  DC       GETNXT
000170 0C05 000D                 CMP.B    #$0D,D5
000174 6704      (0017A          BEQ.S    ENDNAM
000176 10C5                      MOVE.B   D5,(A0)+
000178 60F4      (0016E          BRA.S    NAMLOP
00017A 0028 0080 FFFF    ENDNAM  OR.B     #$80,-1(A0)      SET HI ORDER BIT LAST CHAR
000180 A01D                      DC       SWRITE
                         *
                         * GO FIX DIRECTORY POINTER
                         *
000182 284E                      MOVE.L   A6,A4            MAKE SURE POINTER IS THERE
000184 397C 0002 0022            MOVE.W   #$0002,34(A4)
00018A A01C                      DC       SREAD            GET SECTOR
00018C 204C                      MOVE.L   A4,A0
00018E D1FC 0000 0060            ADD.L    #96,A0           POINT AT SECTOR DATA
000194 4280                      CLR.L    D0
000196 1028 000B                 MOVE.B   11(A0),D0
00019A E148                      LSL.W    #8,D0            HI ORDER DIR PTR
00019C 1028 000C                 MOVE.B   12(A0),D0
0001A0 0640 0020                 ADD.W    #$20,D0          POINTER FOR NEXT DIR ENTRY
0001A4 1140 000C                 MOVE.B   D0,12(A0)
0001A8 E048                      LSR.W    #8,D0
0001AA 1140 000B                 MOVE.B   D0,11(A0)
0001AE A01D                      DC       SWRITE
                         *
                         * NOW CROSS OUT SECTORS IN THE SECTOR MAP
                         *
0001B0 204C                      MOVE.L   A4,A0            GET POINTER
0001B2 D0FC 0060                 ADD      #96,A0           POINT AT DATA AREA IN FCB
0001B6 397C 0001 0022            MOVE.W   #$0001,34(A4)    TRACK 0 SECTOR 1
0001BC A01C                      DC       SREAD
0001BE 323A 017E(0033E           MOVE.W   SCOUNT(PC),D1
0001C2 5241                      ADD.W    #1,D1            FOR FIR SECTOR ADD'L TO DATA SECTORS
                         * FIND FIRST BYTES IN MAP THAT CONTAIN ZEROS
0001C4 2010              L1      MOVE.L   (A0),D0
0001C6 0C40 FFFF                 CMP.W    #$FFFF,D0
0001CA 6604      (001D0          BNE.S    L3
0001CC 5488                      ADD.L    #2,A0
0001CE 60F4      (001C4          BRA      L1
                         * SHIFT 1'S IN UNTIL ALL 1'S OR SECTOR COUNT IS ZERO
```

SCULPTOR



From the world's oldest & largest OS-9 software house!

CUTS PROGRAMMING TIME UP TO 80%

6809/68000-68030 Save 70%

SCULPTOR-a 4GL - Only from S.E. Media at these prices. OS-9 levels one and two (three GIMIX) 6809, all 68XXX OS-9 standard systems. Regular SCULPTOR versions 1.4:6. One of if not the most efficient and easy to develope DBMS type systems running under OS-9! A system of flexible keyed file access that allows extremely fast record and data retrieval, insertion and deletion or other programmed modifications. Access by key or in ascending order, very fast. The system provides automatic menu generation, compilation and report generation. Practically unlimited custom input format and report formatting. A rich set of maintenance and repair utilities. An extremely efficient development environment that cuts most programming approximately 80% in development and debugging! Portable, at source level, to MS-DOS, UNIX and many other languages and systems.

Standard Version: 1.6	6809 - $1295.00
	68000 $1295.00
	68020 $1990.00

Due to a "Special One Time" Purchase, We Are Making This Savings Offer. Quantities Limited!

Once this supply is gone - the price goes back up!

System OS-9: 6809/68000-68030

- Regular ~~$1295.00~~

+ $7.50 S&H USA
Overseas - Shipped Air Mall Collect

ONLY **$295.00**

S.E. MEDIA
POB 849
5900 Cassandra Smith Road
HIXSON, TN 37343 615 842-4601

$AVE - WHILE SUPPLIES LAST!

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

SOFTWARE

!!! Please Specify Your Operating System and Disk Size !!!

SCULPTOR

Full OEM & Dealer Discounts Available!

THE SCULPTOR SYSTEM

Sculptor combines a powerful fourth generation language with an efficient database management system. Programmers currently using traditional languages such as Basic and Cobol will be amazed at what Sculptor does to their productivity. With Sculptor you'll find that what used to take a week can be achieved in just a few hours.

AN ESTABLISHED LEADER

Sculptor was developed by professionals who needed a software development tool with capabilities that were not available in the software market. It was launched in 1981 and since then, with feedback from an ever-increasing customer base, Sculptor has been refined and enhanced to become one of the most adaptable, fast, and above all reliable systems on the market today.

SYSTEM INDEPENDENCE

Sculptor is available on many different machines and for most operating systems, including MS DOS, Unix/Xenix and VMS. The extensive list of supported hardware ranges from small personal computers, through multi-user micros up to large minis and mainframes. Sculptor is constantly being ported to new systems.

APPLICATION PORTABILITY

Mobility of software between different environments is one of Sculptor's major advantages. You can develop applications on a stand-alone PC and – without any alterations to the programs – run them on a large multi-user system. For software writers this means that their products can reach a wider marketplace than ever before. It is this system portability, together with high-speed development, that makes Sculptor so appealing to value added resellers, hardware manufacturers and software developers of all kinds.

SPEED AND EFFICIENCY

Sculptor uses a fast and proven indexing technique which provides instant retrieval of data from even the largest of files. Sculptor's fourth generation language is compiled to a compact intermediate code which executes with impressive speed.

INTERNATIONALLY ACCEPTED

By using a simple configuration utility, Sculptor can present information in the language and format that you require. This makes it an ideal product for software development almost anywhere in the world. Australasia, the Americas and Europe – Sculptor is already at work in over 20 countries.

THE PACKAGE

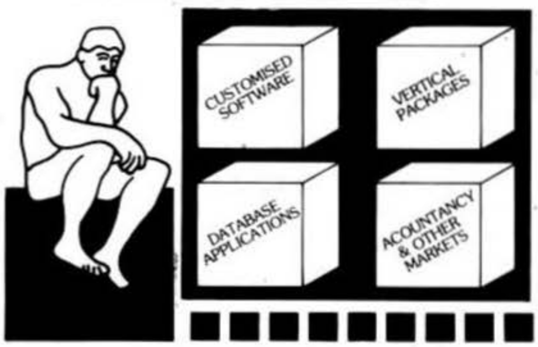
With every development system you receive

- □ A manual that makes sense
- □ A periodic newsletter
- □ Screen form language
- □ Report generator
- □ Menu system
- □ Query facility
- □ Set of utility programs
- □ Sample programs

For resale products, the run-time system is available at a nominal cost.

Facts **Features**

Sculptor for 68020
OS-9 & UniFLEX
$995



DATA DICTIONARY

Each file may have one or more record types described. Fields may have a name, heading, type, size, format and validation list. Field type may be chosen from

- □ alphanumeric
- □ integer
- □ floating point
- □ money
- □ date

DATA FILE STRUCTURE

- □ Packed, fixed length records
- □ Money stored in lowest currency unit
- □ Dates stored as integer day numbers

INDEXING TECHNIQUE

Sculptor maintains a B-tree index for each data file. Program logic allows any numbers of alternative indexes to be coded into one other file.

INPUT DATA VALIDATION

Input data may be validated at three levels

- □ automatic by field type
- □ validation list in data dictionary
- □ programmer coded logic

ARITHMETIC OPERATORS

–	Unary minus
*	Multiplication
/	Division
%	Remainder
+	Addition
–	Subtraction

RELATIONAL OPERATORS

=	Equal to
<	Less than
>	Greater than
< =	Less than or equal to
> =	Greater than or equal to
< >	Not equal to
and	Logical and
or	Logical or
ct	Contains
bw	Begins with

SPECIAL FEATURES

- □ Full date arithmetic
- □ Echo suppression for passwords
- □ Terminal and printer independence
- □ Parameter passing to sub-programs
- □ User definable date format

MAXIMA AND MINIMA

Minimum key length	1 byte
Maximum key length	160 bytes
Minimum record length	3 bytes
Maximum record length	32767 bytes
Maximum fields per record	32767
Maximum records per file	16 million
Maximum files per program	16
Maximum open files	Operating system limit

PROGRAMS

- □ Define record layout
- □ Create new indexed file
- □ Generate standard screen-form program
- □ Generate standard report program
- □ Compile screen-form program
- □ Compile report program
- □ Screen-form program interpreter
- □ Report program interpreter
- □ Menu interpreter

- □ Query facility
- □ Reformat file
- □ Check file integrity
- □ Rebuild index
- □ Alter language and date format
- □ Setup terminal characteristics
- □ Setup printer characteristics

SCREEN-FORM LANGUAGE

- □ Programmer defined options and logic
- □ Multiple files open in one program
- □ Default or programmer processing of exception conditions
- □ Powerful verbs for input, display and file access
- □ Simultaneous display of multiple records
- □ Facility to call sub-programs and operating system commands
- □ Conditional statements
- □ Subroutines
- □ Independent of terminal type

MUSTANG-020 Users - Ask For Your Special Discount!

	*	**	***		*	**	***
MUSTANG-020	***$1,990**	**$398**	**$795**	**PC/XT/AT/MSDOS**	**$695**	**$139**	**$299**
MUSTANG-08	***$1,295**	**$259**	**$495**	**Call or write for prices on the following systems.**			

XENIX SYS III & V, MS-NET, UNIX SYS III & V, ATARI OS-9, 68K, UNOS, ULTRIX/VMS (VAX,REGAL), STRIDE, ALTOS, APRICORT, ARETE, ARMSTRONG, BLEASDALE, CHARLES RIVERS, GMX, CONVERG.TECH, DEC, CIFER, EQUINOX, GOULD, HP, HONEYWELL, IBM, INTEL, MEGADATA, MOTOROLA, NCR, N XDORF, N.STAR, OLIVETTI/AT&T, ICL, PERKINS ELMER, PHILLIPS, PIXEL, PLESSEY, PLEXUS, POSITRON, PRIME, SEQUENT, SIEMENS, SWTPC, SYSTIME, TANDY, TORCH, UNISYS, ZYLOG, ETC.

*** For SPECIAL LOW SCULPTOR prices especially for 6809/68XXX OS-9 Systems - See Special Ad this issue. Remember, "When they are gone the price goes back up as above!"**

... Sculptor Will Run On Over 100 Other Types of Machines ...

... Call for Pricing ...

!!! Please Specify Your Make of Computer and Operating System !!!

* Full Developement Package
** Run Time Only
*** C Key File Library

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CC9 = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343
Telephone: (615) 842-4600 Telex: 5106006630

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

ASSEMBLERS

ASTRUK09 from S.E. Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.

F, S, CCF - $99.95

Macro Assembler for TSC -- The FLEX, SK*DOS STANDARD Assembler.

Special -- CCF $35.00; F, S $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX, SK*DOS.

*FLEX, SK*DOS, CCF, OS-9 $99.00*

Relocating Assembler/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers.

F, S, CCF $150.00

MACE, by Graham Trott from Windrush Micro Systems -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs.

F, S, CCF - $75.00

XMACE -- MACE w/Cross Assembler for 6800/1/2/3/8

F, S, CCF - $98.00

DISASSEMBLERS

SUPER SLEUTH from Computer Systems Consultants Interactive Disassembler; extremely *POWERFUL!* Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems.

Color Computer SS-50 Bus (all w/ A.L. Source)
CCD (32K Req'd) Obj. Only $49.00
F, S, $99.00 - CCF, Obj. Only $50.00 U, $100.00
CCF, w/Source $99.00 O, $101.00
CCO, Obj. Only $50.00
OS9 68K Obj. $100.00 w/Source $200.00

DYNAMITE+ -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00 - CCO, Obj. $ 59.95
F, S, " " $100.00 - O, object only $150.00
U, " " $300.00

CROSS ASSEMBLERS

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/ 40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems.

Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.

*68000 or 6809, FLEX, SK*DOS, CCF, OS-9, UniFLEX*
any object or source each - $50.00
any 3 object or source each - $100.00
Set of ALL object $200.00 - w/source $500.00

XASM Cross Assemblers for FLEX, SK*DOS from S.E. MEDIA -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's.

*Complete set, FLEX, SK*DOS only - $150.00*

CRASMB from LLOYD I/O -- Supports Motorola's, Intel's, Zilog's, and other's CPU syntax for these 8-Bit microprocessors: 6800, 6801, 6303, 6804, 6805, 6809, 6811 (all varieties); 6502, 1802/5, 8048 family, 8051 family, 8080/85, Z8, Z80, and TMS-7000 family. Has MACROS, Local Labels, Label X-REF, Label Length to 30 Chars. Object code formats: Motorola S-Records (text), Intel HEX-Records (text), OS9 (binary), and FLEX, SK*DOS (binary). Written in Assembler ... e.g. Very Fast.

CPU TYPE - Price each:

For:	MOTOROLA	INTEL	OTHER	COMPLETE SET
FLEX9	$150	$150	$150	$399
SK*DOS	$150	$150	$150	$399
OS9/6809	$150	$150	$150	$399
OS9/68K	------	------	------	$432

CRASMB 16.32 from LLOYD I/O -- Supports Motorola's 68000, and has same features as the 8 bit version. OS9/68K Object code Format allows this cross assembler to be used in developing your programs for OS9/68K on your OS9/6809 computer.

*FLEX, SK*DOS, CCF, OS-9/6809 $249.00*

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".

*FLEX, SK*DOS, CCF, OS-9, UniFLEX, 68000 & 6809 with Source $100.00 - without Source $50.00*

X-TALK from S.E. Media - X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTPC 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020. The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020. The X-TALK software is furnished on two disks. One eight inch disk contains S.E. Media modem program C-MODEM (6809) and the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transfered between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also. X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only.

X-TALK Complete (cable, 2 disks)	$99.95
X-TALK Software (2 disks only)	$69.95
X-TALK with CMODEM Source	$149.95

XDATA from S.E. Media - A COMMUNICATION Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc.

U - $299.99

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media

5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

**** Shipping ****
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

PROGRAMMING LANGUAGES

PL/9 from Windrush Micro Systems -- By Graham Trott. A combination Editor Compiler Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide.

F, S, CCF - $198.00

PASC from S.E. Media - A FLEX9, SK*DOS Compiler with a definite Pascal "flavor". Anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. The PASC package comes complete with three sample programs: ED (a syntax or structure editor), EDITOR (a simple, public domain, screen editor) and CHESS (a simple chess program). The PASC package comes complete with source (written in PASC) and documentation.

*FLEX, SK*DOS $95.00*

WHIMSICAL from S.E. MEDIA Now supports *Real Numbers*. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, 6-digit Real Numbers, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer, etc. Run-Time subroutines inserted as called during compilation. *Normally produces 10% less code than PL/9.*

F, S and CCF - $195.00

KANSAS CITY BASIC from S.E. Media - *Basic for Color Computer OS-9* with many new commands and sub-functions added. A full implementation of the IF-THEN-ELSE logic is included, allowing nesting to 255 levels. Strings are supported and a subset of the usual string functions such as LEFT$, RIGHT$, MID$, STRING$, etc. are included. Variables are dynamically allocated. Also included are additional features such as Peek and Poke. A must for any Color Computer user running OS-9.

CoCo OS-9 $39.95

C Compiler from Windrush Micro Systems by James McCosh. Full C for FLEX, SK*DOS except bit-fields, including an Assembler. *Requires the TSC Relocating Assembler if user desires to implement his own Libraries.*

F, S and CCF - $295.00

C Compiler from Introl -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.

*FLEX, SK*DOS, CCF, OS-9 (Level II ONLY), U - $575.00*

PASCAL Compiler from Lucidata -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.

F, S and CCF 5" - $190.00   F, S 8" - $205.00

OmegaSoft PASCAL from Certified Software -- Extended Pascal for systems and real-time programming.

Native 68000/68020 Compiler, $575 for base package, options available. For OS/-9/68000 and PDOS host system.

6809 Cross Compiler (OS-9/68000 host) $700 for complete package.

KBASIC - from S.E. MEDIA -- A "Native Code" BASIC Compiler which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.

*FLEX, SK*DOS, CCF, OS-9 Compiler/Assembler $99.00*

CRUNCH COBOL from S.E. MEDIA -- Supports large subset of ANSII Level 1 *COBOL* with many of the useful Level 2 features. Full FLEX, SK*DOS File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System. *A very popular product.*

*FLEX, SK*DOS, CCF - $99.95*

FORTH from Stearns Electronics -- A CoCo FORTH Programming Language. Tailored to the CoCo! Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!

Color Computer ONLY - $58.95

FORTHBUILDER is a stand-alone target compiler (crosscompiler) for producing custom Forth systems and application programs.

All of the 83-standard defining words and control structures are recognized by FORTHBUILDER.

FORTHBUILDER is designed to behave as much as possible like a resident Forth interpreter/compiler, so that most of the established techniques for writing Forth code can be used without change.

Like compilers for other languages, FORTHBUILDER can operate in "batch mode".

The compiler recognizes and emulates target names defined by CONSTANT or VARIABLE and is readily extended with "compile-time" definitions to emulate specific target words.

FORTHBUILDER is supplied as an executable command file configured for a specific host system and target processor. Object code produced from the accompanying model source code is royalty-free to licensed users.

F, CCF, S - $99.95

EDITORS & WORD PROCESSING

JUST from S.E. Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

• Now supplied as a two disk set:

Disk #1: JUST2.CMD object file,

*JUST2.TXT PL9 source:FLEX, SK*DOS - CC*

Disk #2: JUSTSC object and source in C:

*FLEX, SK*DOS - OS9 - CC*

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files. The C

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CC0 = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media

5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

•• Shipping ••
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX only- F, S & CCF - $49.95
Disk Set (2) - F, S & CCF & OS9 (C version) - $69.95
OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT. with special config section.

*Regular FLEX, SK*DOS $129.50*
• SPECIAL INTRODUCTION OFFER • $79.95
SPECIAL PAT/JUST COMBO (w/source)
*FLEX, SK*DOS $99.95*
OS-9 68K Version $229.00
SPECIAL PAT/JUST COMBO 68K $249.00

Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14.000 plus of free memory! Extra fine for programming as well as text.

*FLEX, SK*DOS $69.95*

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY. INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entire lines, etc. Complete with over 25 different CRT terminal configuration overlays.

*FLEX, CCF. SK*DOS $39.95*

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

*6800 or 6809 FLEX, SK*DOS or SSB DOS, OS-9 - $175.00*

SPELLB "Computer Dictionary" from S.E. Media -- OVER 150,000 words! *Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX, SK*DOS UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F, S and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/SK*DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES 6809 CCF and CCO - $99.95.
F, S or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

NEW PRICES 6809 CCF and CCO - $69.95.
F, S or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

NEW PRICES 6809 CCF and CCO - $59.95.
F, S or O - $79.95, U - $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!

F, S or O - $329.95, U - $549.95
O, 68000 $695.00

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems
FOR 6809 FLEX-SK*DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character file names! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface. Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CC0 = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

ITS EASY TO USEI

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...)

The possibilities are unlimited...

FOR 6809 FLEX-SK*DOS(5/8") $249.95

UTILITIES

Basic09 XRef from S.E. Media -- This Basic09 Cross Reference Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a Program List Utility which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or RunB.

O & CCO obj. only -- $39.95; w/ Source - $79.95

BTree Routines - Complete set of routines to allow simple implementation of keyed files - *for your programs* - running under Basic09. A real time saver and should be a part of every serious programmers tool-box.

O & CCO obj. only - $89.95

Lucidata PASCAL UTILITIES (Requires Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary - unlimited nesting.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, S, CCF --- EACH 5" - $40.00, 8" - $50.00

DUB from S.E. Media – A UniFLEX BASIC decompiler Re-Create a Source Listing from UniFLEX Compiled basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic.

U - $219.95

LOW COST PROGRAM KITS from Southeast Media The following kits are available for FLEX, SK*DOS on either 5" or 8" Disk.

1. **BASIC TOOL-CHEST $29.95**
   BLISTER.CMD: pretty printer
   LINEXREF.BAS: line cross-referencer
   REMPAC.BAS, SPCPAC.BAS, COMPAC.BAS:
   remove superfluous code
   STRIP.BAS: superfluous line-numbers stripper
2. **FLEX, SK*DOS UTILITIES KIT $39.99**
   CATS. CMD: alphabetically-sorted directory listing
   CATD.CMD: date-sorted directory listing
   COPYSORT.CMD: file copy, alphabetically
   COPYDATE.CMD: file copy, by date-order
   FILEDATE.CMD: change file creation date
   INFO.CMD (& INFOGMX.CMD): tells disk attributes & contents
   RELINK.CMD (& RELINK82): re-orders fragmented free chain
   RESQ.CMD: undeletes (recovers) a deleted file
   SECTORS.CMD: show sector order in free chain
   XL.CMD: super text lister
3. **ASSEMBLERS/DISASSEMBLERS UTILITIES $39.95**
   LINEFEED.CMD: 'modularise' disassembler output
   MATH.CMD: decimal, hex, binary, octal conversions
   & tables
   SKIP.CMD: column stripper
4. **WORD - PROCESSOR SUPPORT UTILITIES $49.95**
   FULLSTOP.CMD: checks for capitalization
   BSTYCIT.BAS (.BAC): Stylo to dot-matrix printer
   NECPRINT.CMD: Stylo to dot-matrix printer filter code
5. **UTILITIES FOR INDEXING $49.95**
   MENU.BAS: selects required program from list below
   INDEX.BAC: word index
   PHRASES.BAC: phrase index
   CONTENT.BAC: table of contents
   INDXSORT.BAC: fast alphabetic sort routine
   FORMATER.BAC: produces a 2-column formatted index
   APPEND.BAC: append any number of files
   CHAR.BIN: line reader

BASIC09 TOOLS consist of 21 subroutines for Basic09.
6 were written in C Language and the remainder in assembly.
All the routines are compiled down to native machine code which makes them fast and compact.

1. **CFILL** -- fills a string with characters
2. **DPEEK** -- Double peek
3. **DPOKE** -- Double poke
4. **FPOS** -- Current file position
5. **FSIZE** -- File size
6. **FTRIM** -- removes leading spaces from a string
7. **GETPR** -- returns the current process ID
8. **GETOPT** -- gets 32 byte option section
9. **GETUSR** -- gets the user ID
10. **GTIME** -- gets the time
11. **INSERT** -- insert a string into another
12. **LOWER** -- converts a string into lowercase
13. **READY** -- Checks for available input
14. **SETPRIOR** -- changes a process priority
15. **SETUSR** -- changes the user ID
16. **SETOPT** -- set 32 byte option packet
17. **STIME** -- sets the time
18. **SPACE** -- adds spaces to a string
19. **SWAP** -- swaps any two variables
20. **SYSCALL** -- system call
21. **UPPER** -- converts a string to uppercase

For OS-9 - $44.95 - Includes Source Code
Limited Special - $19.95

SOFTOOLS

The following programs are included in object form for immediate application. PL/9 source code available for customization.

READ-ME Complete instructions for initial set-up and operation. Can even be printed out with the included text processor.

CONFIG one time system configuration.

CHANGE changes words, characters, etc. globally to any text type file.

CLEANTXT converts text files to standard FLEX, SK*DOS files.

COMMON compare two text files and reports differences.

COMPARE another check file that reports mis-matched lines.

CONCAT similar to FLEX, SK*DOS append but can also list files to screen.

DOCUMENT for PL/9 source files. Very useful in examining parameter passing aspects of procedures.

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CC9 = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media

5900 Cassandra Smith Rd. - Hixson, Tn. 37343

mastercard

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

ECHO echos to either screen or file.

FIND an improved *find* command with "*pattern*" matching and wildcards. Very useful.

HEX dumps files in both hex and ASCII.

INCLUDE a file copy program that will accept "includes" of other disk files.

KWIC allows rotating each word, on each line to the beginning. Very useful in a sort program, etc.

LISTDIR a directory listing program. Not super, but better than CAT.

MEMSORT a high-speed text file sorter. Up to 10 fields may be sorted. Very fast. Very useful.

MULTICOL width of page, number of columns may be specif d. A MUST!

PAGE similar to LIST but allows for a page header, page width and depth. Adjust for CRT screen or printer as set up by CONFIG. A very smart print driver. Allows printer control commands.

REMOVE a fast file deleter. Careful, no prompts issued. Zap, and its gone!

SCREEN a screen listing utility. Word wraps text to fit screen. Screen depth may be altered at run time.

SORT a super version of MEMSORT. Ascending/descending order, up to 10 keys, case over-ride, sort on n° word and sort on characters if file is small enough, sorts in RAM. If large file, sort is constrained to size of your largest disk capacity.

TPROC a small but nice text formatter. This is a complete formatter and has functions not found in other formatters.

TRANSLIT sorts a file by x keyfields. Checks for duplications. Up to 10 key files may be used.

UNROTATE used with KWIC this program reads an input file and unfolds it a line at a time. If the file has been sorted each word will be presented in sequence.

WC a word count utility. Can count words, characters or lines.

NOTE: this set of utilities consists of 6 5-1/4" disks or 2 8" disks, w/ source (PL9). 3 5-1/4" disks or 1 8" disk w/o source.
Complete set SPECIAL INTRO PRICE:
5-1/4" w/source FLEX - SK*DOS - $129.95
w/o source - $79.95
8" w/source - $79.95 - w/o source $49.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F, S and CCF, U - $25.00, w/ Source - $50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/ Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!
Levels I & II only - OS-9 $69.95

DISK UTILITIES

OS-9 VDisk from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
Level I OS-9 obj. $79.95; w/ Source $149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX, SK*DOS Format so it can be used normally by F EX, SK*DOS; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX, SK*DOS Directory, Delete FLEX, SK*DOS Files, Copy both directions, etc. FLEX, SK*DOS users use the special disk just like any other FLEX, SK*DOS disk
O - 6809/68000 $79.95

LSORT from S.E. Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full set of comments and errors messages.
OS-9 $85.00

HIER from S.E. Media - *HIER is a modern hierarchal storage system for users under FLEX, SK*DOS.* It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size. **Using HIER a regular (any) FLEX, SK*DOS disk (8 - 5 - hard disk)** can have sub directories. By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modern day solution that all current large systems use. **Each directory looks to FLEX, SK*DOS like a regular file,** except they have the extension '.DIR'. A full set of directory handling programs are included, making the operation of HIER simple and straightforward. A special install package is included to install HIER to your particular version of F EX, SK*DOS. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!
*FLEX - SK*DOS $79.95*

COPYMULT from S.E. Media -- Copy LARGE Disks to several smaller dis s. FLEX, SK*DOS utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting dis ettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORE.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

*Completely documented Assembly Language Source files included. ALL 4 Programs (FLEX, SK*DOS, 8" or 5") $99.50*

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CC0 = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

** Shipping **
Add 3% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

COPYCAT from Lucidata -- *Pascal NOT required.* Allows reading TSC Mini-FLEX, SK*DOS, SSB DOS68, and Digital Research CP/M Disks while operating under SK*DOS , FLEX1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.

F, S and CCF 5" - $50.00 F, S 8" - $65.00

VIRTUAL TERMINAL from S.E. Media - Allows one terminal to do the work of several. The user may start as many as eight tasks on one terminal, under *VIRTUAL TERMINAL* and switch back and forth between tasks at will. No need to exit each one; just jump back and forth. Complete with configuration program. The best way to keep up with those background programs.

6809 O & CCO - obj. only - $49.95

FLEX, SK*DOS DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX, SK*DOS Utilities for every FLEX, SK*DOS Users Toolbox: Copy a File with CRC Errors; Test Disk for errors; Compare two Disks; a fast Disk Backup Program; Edit Disk Sectors; Linearize Free-Chain on the Disk; print Disk Identification; and Sort and Replace the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A BASIC Resequencer with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs Compare, Merge, or Generate Updates between two BASIC Programs, check BASIC Sequence Numbers, compare two unsequenced files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in TSC BASIC, XBASIC, and PRECOMPILER BASIC Programs.

ALL Utilities include Source (either BASIC or A.L. Source Code).

F, S and CCF - $50.00

BASIC Utilities ONLY for UniFLEX -- $30.00

MS-DOS-FLEX Transfer Utilities to OS-9 For 68XXX and CoCo* OS-9 Systems Now READ - WRITE - DIR - DUMP - EXPLORE FLEX & MS-DOS Disk. These Utilities come with a rich set of options allowing the transfer of text type files from/to FLEX & MS-DOS disks. *CoCo systems require the D.P. Johnson SDISK utilities and OS-9 and two drives of which one must be a "host" floppy.

**CoCo Version: $69.95 68XXX Version $99.95*

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.

F, S and CCF, U - $50.00, w/ Source - $100.00

DYNACALC -- Electronic Spread Sheet for the 6809 and 68000.

F, S, OS-9 and SPECIAL CCF - $200.00, U - $395.00

OS-9 68K - $595.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.

F, S and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F, S and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from S.E. Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F, S - $59.95, U - $89.95

GAMES

RAPIER - 6809 Chess Program from S.E. Media -- Requires FLEX, SK*DOS and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, *estimated USCF Rating 1600+ (better than most 'club' players at higher levels)*

F, S and CCF - $79.95

NEW

MS-DOS/FLEX Transfer Utilities For 68XXX and CoCo* OS-9 Systems. Now Read, Write, DIR, Dump and Explore FLEX & MS-DOS Disks. Supplied with a rich set of options to explore and transfer text type files from/to FLEX and MS-DOS disks. *CoCo OS-9 requires SDISK utilities & two floppy drives.

CCO $69.95 68XXX OS-9 $99.95

NOTE: Changes

1. Price increase for SCULPTOR, see advertising front of this catalog and other ad in this issue. Special price for 68 Micro Journal readers.
2. Lower price for BASIC09 TOOLS, see Utilities section.
3. New MS-DOS & FLEX to OS-9 Utilities, see above.

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

**** Shipping ****
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

```
0001D0 E280               L3       ASR.L    #1,D0
0001D2 5341                        SUB.W    #1,D1               DECREMENT SECTOR COUNT
0001D4 4A41                        TST.W    D1
000106 670C      (001E4            BEQ.S    L4                  DONE
0001D8 DC40 FFFF                   CMP.W    #$FFFF,D0
0001DC 66F2      (001D0            BNE      L3
                          * IF RAN OUT OF 1'S BUT STILL HAVE SECTORS TO CROSS OUT
0001DE 2080                        MOVE.L   D0,(A0)
0001E0 5488                        ADD.L    #2,A0
0001E2 60E0      (001C4            BRA      L1
0001E4 2080               L4       MOVE.L   D0,(A0)
0001E6 A010                        DC       SWRITE
0001E8 A01E                        DC       WARMST              WE'RE DONE!
                          *
                          * SUBROUTINE TO SAVE FIR TRACK AND SECTOR
                          *
0001EA 3828 001E          GETLST   MOVE.W   30(A0),D4           DTTS
0001EE 5C44                        ADD.W    #6,04
0001F0 E944                        ASL.W    #4,D4               0000TTS0
0001F2 E80C                        LSR.B    #4,D4               0000TT0S
0001F4 47FA 0144(0033A             LEA      1ASTS(PC),A3
0001F8 3684                        MOVE.W   D4,(A3)
0001FA 4E75                        RTS
                          *
                          * PRINT A NULL TERMINATED STRING
                          *
0001FC 2648               PSTR     MOVE.L   A0,A3               USE A3 FOR STRING POINTER
0001FE 181B               PLOOP    MOVE.B   (A3)+,04
000200 0C04 0000                   CMP.B    #0,D4
000204 6704      (0020A            BEQ.S    PEND
000206 A033                        DC       PUTCH
000208 60F4      (001FE            BRA.S    PLOOP
00020A 4E75               PEND     RTS
                          *
                          * ADVANCE TO NEXT SECTOR ON DISK
                          * A4 TO POINT AT FCB ON ENTRY
                          *
00020C 522C 0023          NEWTS    ADD.B    #1,35(A4)
000210 0C2C 0010 0023              CMP.B    #$10,35(A4)
000216 6608      (00220            BNE.S    DONETS
000218 422C 0023                   CLR.B    35(A4)
00021C 522C 0022                   AD0.B    #1,34(A4)
000220 4E75               DONETS   RTS

000222 A037               ERROR    DC       PERROR
000224 A01E                        DC       WARMST

000226 49FA 0006(0022E    HELP     LEA      HLPMSG(PC),A4
00022A A035                        DC       PSTRNG
00022C A01E                        DC       WARMST

00022E 5379 6E74 6178     HLPMSG   DC.B     "Syntax: COS9 SKFIlENAME DR OSFIlENAME",$0D,$0A
000255 434F 5339 2063              DC.B     "COS9 copies an SK*DOS text file to an OS9 format",$0D,$0A
000287 6469 736B 2E20              DC.B     "disk.  SKFILENAME defaults to working drive and",$0D,$0A
0002B8 2E54 5854 2065              DC.B     ".TXT extension.  DR is logical drive number of",$0D,$0A
0002E8 7468 6520 6472              DC.B     "the drive containing the OS9 disk.  OSFIlENAME",$0D,$0A
000318 6973 2074 6865              DC.B     "is the desired OS9 filename.",$0D,$0A,$04
                          *
                          * VARIABlES HERE
                          *
000338                    FIRSEC   DS.W     1
00033A                    LASTS    DS.W     1
00033C                    OIRTS    DS.W     1
00033E                    SCOUNT   DS.W     1                   COUNT OF SECTORS WRITTEN TO DISK
000340                    INFCB    DS.B     608

                                   END      START
 0 ERRORS DETECTED

EOF
```

The Macintosh™ Section

Reserved as

A place for your thoughts

And ours.......

Mac - Watch

By: James E. Law
1806 Rock Bluff Road
Hixson, TN 37343

ReadySetGo Revisited

Several months ago, I reviewed Ready,Set,Go! 3.0 and found it, in spite of some minor irritants, to be an easy-to-use and powerful desktop publishing tool. Since then, Revision 4.0 has been released. This does not change the way Ready,Set,Go! works, but contains so many new features and enhancements that it must be considered a major upgrade. The purpose of this article will be to briefly tell you about the changes to Ready,Set,Go! introduced in Rev. 4.

Getting Ready

The first thing you notice when starting Ready,Set,Go! 4.0 is that the working window has been rearranged by removing the tool bar from the left of the screen. These tools have been repositioned along the top of the screen or included in various pull down menus. This makes for a cleaner working area which is 13% bigger than the Ready,Set,Go! 3.0 window.

Ready,Set,Go! 4.0 makes it easier to work with larger documents. It allows you to set up any size document up to 99" by 99." Also, a tiling feature is provided for printing ImageWriter and LaserWriter copies of very large documents. RSG4 accomplishes this by dividing the overall image into overlapping 8-1/2" by 11" tiles. You can control the amount of the overlap, and registration marks are provided for precisely lining up the sheets.

I was glad to note that a grabber tool is provided in Ready,Set,Go! 4.0 for moving around without having to use the scroll bars. The lack of such a tool in Ready,Set,Go! 3.0 was often a source of frustration as I tried to rapidly navigate to the desired spot on a large page.

In Ready,Set,Go! 3.0 the ability to view facing pages was helpful in achieving a balanced layout. Unfortunately, no editing could be done while in this viewing mode. In Ready,Set,Go! 4.0, however, the facing pages view is interactive. This makes it easy to design and adjust graphics which stretch across both pages.

Adjusting the Fine Print

One of the things that impressed me most about Ready,Set,Go! 3.0 was the extensive control provided over the aesthetics of text. Ready,Set,Go! 4.0, however, increases this control significantly. For example, Ready,Set,Go! 3.0 allowed great flexibility in setting letter, line, and paragraph spacing. In addition to those parameters, Ready,Set,Go! 4.0 allows you to control the spacing between words. The handling of tabs in Ready,Set,Go! 3.0 was a source of confusion and complaints. This feature has been revised in Ready,Set,Go! 4.0 to make it simpler and more intuitive.

In addition to all the normal text styles (e.g., plain, bold, outline, etc.), Ready,Set,Go! 4.0 makes available condensed and expanded text, over-struck text, and reversed type.

If you are a "detail" person, you'll love the ability to set hyphenation specifications in Ready,Set,Go! 4.0 You can set the minimum size word to be hyphenated, the minimum number of characters before and after the hyphen, and the maximum number of consecutive hyphens that may occur. You can also indicate whether the last word in a paragraph or capitalized words can be hyphenated.

With all the flexibility provided by this program to specify in detail the attributes of text, creation of document with unclear or unattractive typography will definitely not be Ready,Set,Go! 4.0's fault.

It's Got Style

Ready,Set,Go! 4.0 allows users to save time on repetitive setups by preparing style sheets. For example, separate style sheets might be set up for the body of a document, for minor headings, and for major headings. Macros can be set up which enable a style sheet to be called by holding down the COMMAND key and typing "H" then the assigned character. Style sheets cover font type, style, and size; left, right, centered, or justified text; tab settings; indents; word, line, and paragraph spacing, and whether hyphenation is on or off. Style sheets can be saved, imported, duplicated, modified, or deleted.

Another way of making universal changes to font type, style, or size is to use the search and replace function. You could, for example, automatically change all underlined words to bold text. Similarly, all Geneva 12 text could be automaticallychanged to Bookman 10. This function could be of great help to those making last minute format changes to large documents.

Handling Graphics

Ready,Set,Go! 4.0 imports graphics in TIFF and EPS format in addition to the PICT and bitmapped formats handled by Ready,Set,Go! 3.0. Also, it provides greatly increased flexibility in managing the visual relationship between text and adjacent text. When graphics is placed within a text block, "run around" can be turned off in which case the text flows right over the graphics. Alternately, the graphics block (an invisible rectangle around the graphics) may be set to repel the text. A new option presented by Revision 4 is to have the text repelled only by the graphic object. In this case, the text neatly follows the outline of an irregular shaped graphic. This can be used to create some really striking layouts! The distance the text is repelled by the graphics or the graphic block is adjustable.

But what About theManual

If you read reviews about Ready,Set,Go! 3.0, you know that no one liked the manual. It was a skimpy publication which was more like a magazine than a users manual. The Ready,Set,Go! 4.0 package largely fixes this problem by including a 218-page manual along with a very helpful book on layout. The users manual seems complete and clearly written.

Fine Tuning

Ready,Set,Go! 4.0 has some compatibility problems with the printer driver included with the Apple MultiFinder. This occurred because tight schedule forced Letraset to test Ready,Set,Go! 4.0 against a prerelease version of the MuliFinder package. Apple then changed the coding of the printer driver before final release. A maintenance revision 4.0a has been prepared to solve this problem and is scheduled to be shipped free to all registered owners in late Feburary.

I reviewed a beta copy of Ready,Set,Go! 4.0a and verified that it is completely compatable with the MultiFinder printer driver. Additionally , it includes the following enhancements:

1. The readibility of text in the 200% view has been improved.
2. Double clicking any text block, graphics block, or graphics element brings up the related specification block.
3. Ready,Set,Go! 4.0 supports additional versions of TIFF.
4. The highly compact RIFF files created by Letraset' s new photo retouching program called ImageStudio can be read directly.

In Conclusion. . .

With Ready,Set,Go! 4.0, Letraset clearly demonstrates their commitment to being taken seriously in the Macintosh desktop publishing nitch. This is a powerful product that will well serve the needs of most users. The "big name" competition had better keep on their toes or Ready,Set,Go! will leave them behind.

EOF

Logically Speaking

Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

The Mathematical Design of Digital Control Circuits

By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

SOLUTION TO TEST NINE

Let's talk our way through this one to show how the flow-table is developed. First, we'll let the outputs Z1, T1 and T2 represent the alarm, a 2-minute timer and a 3-minute timer respectively.

$X_1X_2T_1T_2$	0000	1000	0010	0110	0100	0011	0111
1	1	2		1	1		
	000	000		000	000		
2	2	2	2	1	3	2	3
	011	000	011	010	φ00	011	φ11
3					3		3
					100		111

As usual, we'll commence at address 0000.1 with a stable all-zero output-state. Operation of X1 moves us to address 1000.2 (with all outputs held at 0), and its subsequent release slides us to 0000.2, at which time we'll energise both T1 and T2. In this location we're now waiting to see whether T1's timing-out or X2 will occur first. We'll assume that X2 operates, which moves us to address 0100.3 with the alarm Z1 operated. On the elbow, 0100.2, we'll make Z1 a phi AND DE-ENERGISE BOTH T1 AND T2. It's important at least to de-energise T1, in case it should time out while we're moving vertically downwards to 0100.3 and thus enter into a critical race with this movement as it tries to slide us horizontally into a new column. We're now in a stable state with the alarm ON.

On the other hand, T1 may time out first (2-minutes elapsed), in which case we'll move from 0000.2 to 0010.2, where we'll sit in a stable state, waiting to see whether T2's timing-out or X2 will occur first. Again let's assume that X2 occurs first, sliding us first to 0110.2, where we'll de-energise T2 in case it should choose to time out at this critical moment, and then vertically to 0110.1. Here's where we'll de-energise T1, rather than directly in 0110.2, to avoid a critical race between the dropping-out of T1's contact and the relay moving us vertically. The de-energising of T1, of course, moves us sideways to address 0100.1, with all Zs OFF.

Back to address 0010.2 again, to consider the possibility of T2's timing-out first. This would slide us sideways to address 0011.2, where we'll wait indefinitely for X2 to arrive and send us to address 0111.3, thus activating the alarm.

Only one merger is possible - that of rows 2 and 3, to give us a 2-row merged flow-table. As this requires only one relay for its implementation, we'll code row 1 with a red 0 and row 2 with a red 1, and allocate the values 16, 8, 4 and 2 to X1, X2, T1 and T2.

All Box-Bs are then completed, and an intermediate decoding-table prepared, after which it only remains to carry out the actual decoding and draw the circuit diagram from the Boolean expressions. It would probably be a good idea for you to try the decodings with the card-decoder to be described next, and check off your results against the tables you produced manually.

$x_1x_2T_1T_2$	0000	0	1000	16	0010	4	0110	12	0100	8	0011	6	0111	14
1	1	0	2	16			1	12	1	8				
0	000		000				000		000					
2	2	1	2	17	2	5	1	13	2	9	2	7	2	15
1	011		000		011		010		100		011		111	

	0	1	5	7	8	9	12	13	15	16	17
Y_1		1	1	1		1			1	1	1
Z_1						1			1		
T_1		1	1	1				1	1		
T_2		1	1	1					1		

Y_1

1	5	7	9	15	16	17	16 X_1	8 X_2	4 T_1	2 T_2	1 y_1	ϕ
x			~~8~~			16	ϕ	ϕ	0	ϕ	1	25, 3, 11, 19, 27
4	x	2				4	ϕ	0	ϕ	ϕ	1	21, 3, 19, 23 ✓
		8		x			ϕ	ϕ	ϕ	1	ϕ	31, 23, 11, 3, 27, 19 + 8 others
					x	1	1	ϕ	ϕ	ϕ	ϕ	24, 20, 28, 18, 26, 22, 30 + 7 more
	x	2					ϕ	0	1	ϕ	ϕ	4, 21, 20, 6, 23, 22

Z_1

9	15	16 X_1	8 X_2	4 T_1	2 T_2	1 y_1	ϕ
x		ϕ	1	0	ϕ	1	25, 11, 27
	x	ϕ	1	ϕ	1	ϕ	31, 11, 27 +

T_1

1	5	7	13	15	16 X_1	8 X_2	4 T_1	2 T_2	1 y_1	ϕ
x	4	2			0	0	ϕ	ϕ	1	3
	8	2	x	2	ϕ	ϕ	1	ϕ	1	29, 21, 31, 23

T_2

1	5	7	15	16 X_1	8 X_2	4 T_1	2 T_2	1 y_1	ϕ
x	4	2		0	0	ϕ	ϕ	1	3
		8	x	ϕ	ϕ	ϕ	1	ϕ	31, 23, 11, 3, 27, 19 +

In case you've forgotten, here's how we figure out the minterm numbers for Y1, where rows 1 and 2 correspond to a 2-unit state-diagram. Note that to the left of the flow-table there's a red 1 in row 2. The red 1, which symbolises Y1's energisation, indicates that Y1 will be energised wherever there's a black 2 (because we're looking at row 2) in the flow-table itself. So we go through the table looking for black 2s, and when we find one we note the corresponding red minterm-number in Box-B and put a 1 for Y1 under this minterm-number in the intermediate table. Got it? OK, now we'll proceed with actual decoding.

Notice that in the decoding for Y1 we've created an auxiliary decoding row because we were blocked by y1 in row 2. However, our original row (ie 2) allows a factoring of y1 with row 1, so we'll stick with row 2 as the auxiliary row allows no factoring.

The final Boolean expressions are therefore :

$$Y1 = T1'.y1 + X2'.y1 + T2 + X1 = X1 + T2 + y1(T1' + X2')$$
$$Z1 = X2.T1'.y1 + X2.T2 \qquad = X2(T1'.y1 + T2)$$
$$T1 = X1'.X2'.y1 + T1.y1 \qquad = y1(X1'.X2' + T1)$$
$$T2 = X1'.X2'.y1 + T2$$

There are other ways to implement the specs, of course, such as making T2 a 1-minute timer. With this approach, only T1 would be energised at address 0000.2, and T2 activated when T1 times out.

Mile 11 - heading for Mile 12

BEFORE WE PROCEED

In all our worked examples, you'll note that I've taken through to completion only the one approach to a problem. In normal practice, I'd scan for "hazards" (re-read an earlier lesson for an explanation) and/or try decoding the 0s and phis (and then complementing the expressions) to see if there's a better circuit. However, if I did this with each and every example I present to you, we'd get slowed down too much, so I'll leave that part up to you! Now for

CARD-DECODING OF DECIMAL MINTERMS

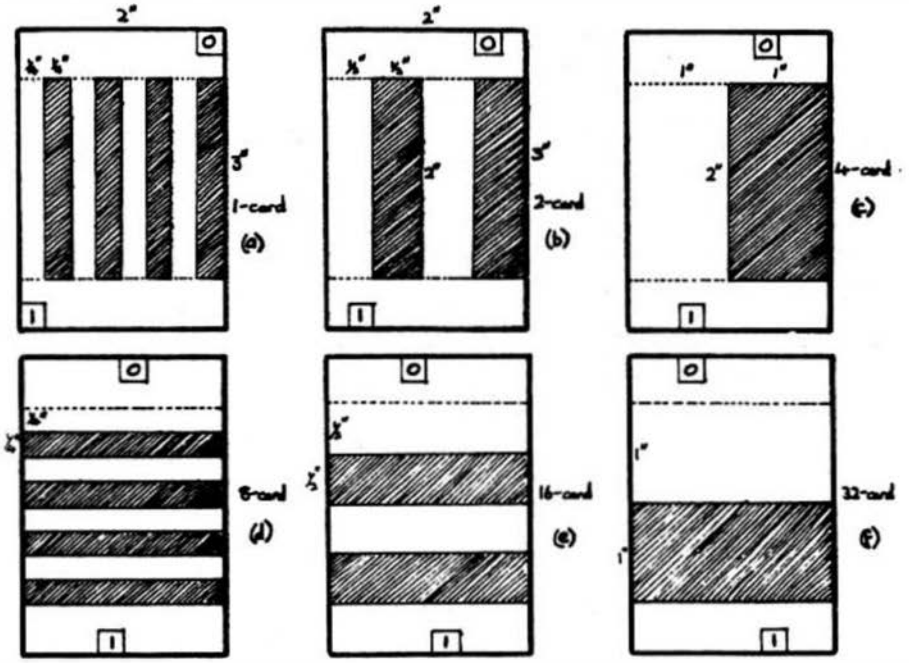
You'll find, when you become involved in designing more complex circuits, that as the number of variables increases, and also the number of Ys and Zs to be decoded, the task of decoding the minterm-numbers becomes quite heavy, and the possibility of error becomes greater too. This means that all your calculations will have to be checked, as it's much better to find an error at this stage than to build your machine, and then find that it doesn't behave the way it should because of some small(?) slip in the decoding.

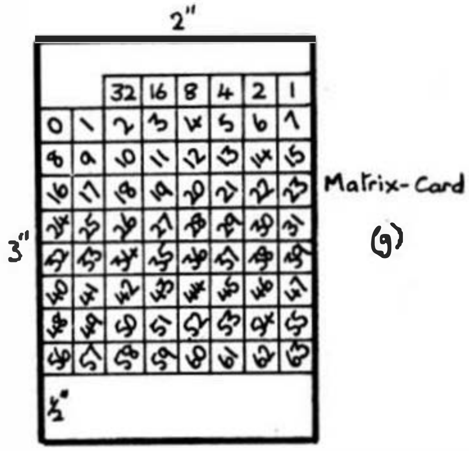
I can state from painful experience, too, that there's nothing more frustrating than, let us say, trying to eliminate the ninth variable in a ten or eleven variable decoding, to find that '2' will "go with" 127 of the minterm-numbers only to end up being blocked on the 128th. We're faced, in decodings of this magnitude, with the horrible choice of checking through first to see if this particular variable CAN be converted to phi, and if so to repeat all our calculations a second time inserting, say, 2s under the minterms, OR putting the 2s in as we go along and then having to go back and erase them all should we get blocked somewhere along the way. Either way, it's not an easy task!

There just HAD to be a better way than this, so several years ago, when I was involved in decoding more than thirty Ys and Zs in ten variables (with each device's decoding producing about 12 rows and taking approximately two-and-a-half hours to carry out), I designed a set of punched cards which reduced the decoding time for each output to about 3 or 4 minutes. Further, when I checked over the expressions I'd already decoded I was amazed at the number of errors which had crept in. In some cases I found that I'd gone to the trouble of writing 2 under maybe 128 minterms, only (in my joy at finding that it "went") to forget to change the variable concerned into a phi. All that work absolutely wasted!!**??!!

Since then I've constructed a smaller 6-variable decoder which is extremely compact, made from thin sheets of clear acetate, with strips of black masking-tape (or draughting-tape) to mask out the cross-hatched bars indicated in the drawings of Diagram 48. My original 10-variable set was made of sheets of opaque plastic and the clear portions (corresponding to those in the 2" x 2" central square of my later set) were reamed out instead.

Anyway, there are six cards altogether (one for each variable) as shown in Diagram 48a through 48f, plus a backing matrix of minterm-numbers shown in 48g.





All cards are 2 inches by 3 inches, and all masking strips are 2 inches long. You'll notice that the width of the space between the strips is the same as that of the strip itself for all cards. The cards are arranged so that when the figure '0' appears at the top of the card, there's a clear strip down the left-hand side (or along the top) of the centre 2 x 2 square. If the card is rotated so that a '1' appears at the top, the opposite is true, that is, a dark strip is now down the left-hand side or along the top. Note too that these digits 0 and 1 move inwards by 1/4 inch as we progress through the series of cards (a) through (f). ALL DIAGRAMS ARE DRAWN TO SCALE.

The cards should be made from a material thin enough and clear enough to see reasonably through six layers, but thick enough to retain a certain amount of rigidity.

Having made these six cards, the final step is to draw 48g on a piece of paper or thin card, and over it to paste a clear sheet of 2 x 3 acetate.

GETTING THE FEEL OF THE CARDS

Now, to begin with, let's stack the cards together with the 1-card on the bottom and the 32-card on the top, and with all 0s at the top of the cards. Along the top of the cards you should now read the binary number 000000, and to our great surprise, if this stack is placed on top of the backing-matrix, what do we see through the single clear 1/4 x 1/4 window in the central 2 x 2 section? Why, the decimal equivalent of this number, that is 0. Let's rotate some of the cards, so that at the top we see different binary numbers, let's say 101101, and place them on top of the matrix. Again we see the decimal equivalent, namely 45. Try a few more for yourself, and then we'll learn how to use them to perform the decoding operation for us.

USING THE CARDS TO DECODE A SET OF MINTERMS

Well, I think we're now ready to try a simple decoding, so for a trial run let's do Press Z1's decoding of Diagram 47b. Just as before, we draw up the framework of the decoding-table, and we're set to go, but first, as there are only four variables involved, we'll put to one side the top two cards, namely the 32-card and the 16-card.

We insert our usual 'x' under minterm-1 and write 0001 to the right, which is the number to which we must set up our four remaining cards. Having done so, our next step is to take the matrix-card and (with a felt-tip pin or a pencil capable of writing on plastic) we put a bright red 'x' or medium-sized dot over each decimal number corresponding to an UNDESIRED minterm in the intermediate decoding table for Z1, ie 0, 2, 3, 7, 10, 11 and 15. These correspond to 0s. Note that we do NOT mark phis.

Now, if we place the cards over the matrix, we should see the number 1 through the window. So ... holding the cards and the matrix in a neatly stacked pile, we remove the top 8-card and look at the additional window which has been created. We see here the figure 9 (compare with the original decoding table in Diagram 47b), because 1 + 8 = 9 is available. We therefore place the 8-card to one side, and remove the 4-card. Two more numbers pop into view, namely 5 and 13, so the 4-card can be discarded. Of course, the 5 and 13 correspond to adding 4 to each of our already existing minterms 1 and 9.

Proceeding, we remove the 2-card and what is this? We see a lot of red 'x's through the additional windows thus created. This means that we can't discard this card, so we re-insert it in the pack IMMEDIATELY ABOVE THE MATRIX

CARD. This would also be done even if only one red 'x' were visible, as we find to be the case when we try removing the 1-card from the pack.

And so we're left with two cards, whose binary headings, 0 and 1, occur over the bit-positions 2 and 1 at the top of the matrix. We therefore convert to phis in our decoding-table the variables occurring in bit-positions 8 and 4 (corresponding to our discarded cards), and place a tick under those minterms whose numbers are visible through the windows in our decoding cards.

That's one decoding row done, so let's carry on with row 2 and at the same time polish up our procedure slightly. Beginning with an 'x' under minterm-4 (the next minterm not already covered) and writing 0100 to the right, we now set up our four cards to read 0100, and place them on top of the matrix, verifying that 4 does in fact show through the single window opened up. Removing the 8-card does not show red, so we discard it. On removing the 4-card we do see red, but before re-inserting it ON TOP OF THE MATRIX we'll check to see whether one of the red 'x's is due to the base-minterm 4 itself. This is done by noting that we're trying to discard the 4-card, currently showing 1 at the top, ie, we're trying to SUBTRACT 4 FROM THE BASE 4. As 4 - 4 = 0, and we see a red 0 through one of the windows, it means that we can't even begin a run, so we re-insert the card. On the other hand, if the 0 seen through the windows were NOT red, it would indicate that we could start a run, but would get blocked along the way by one of the other red (forbidden) numbers.

When we lift the 2-card no red shows, which means, of course, that we can discard it, but when we remove the 1-card we see red again, NOT DUE TO THE BASE-MINTERM, as our check shows that 4 + 1 = 5 is a clear number. Don't forget, we ADD 1 to the base here because we're trying to convert an existing 0 card into a phi, which can only be done by superimposing a 1 over the 0. In this instance, as we started a run (because 5 is clear) we put our usual tick to the right of the row to indicate a non-essential prime-implicant, and a dot over the 1-variable to indicate a block at this point. Again we find ourselves left with two cards. the 4-card and the 1-card, and we therefore convert the discarded variables in row 2 to phis, and put a tick under minterm-12. With the automatic decoder we don't bother to record the phi-minterms to the right of the table as we do with manual decoding.

As we were blocked by the 1-variable in row 2, we'll create an auxiliary decoding row commencing once more on minterm-4, and stack up the four cards just as before. But this time we'll remove the 1-card FIRST, and then proceed systematically from the top of the stack downwards, to end up with the 4-card and the 2-card. Then we convert to phis the discarded 1 and 8 variables, and place ticks under minterms 5, 12 and 13. Along the way, we'll have placed a tick to the right of the row to indicate a non-essential prime-implicant the moment we found ourselves blocked by the 2-variable, and also put a dot over this variable in row 3.

Before doing another device's decoding it goes without saying that we commence by wiping out our matrix, and marking red 'x's for the new set of forbidden (or 0) states.

Also, if we're going to do a 0-decoding in order to eliminate a hazard or merely to see if it produces a "better" decoding, the 1s become forbidden, and we mark them in red instead. As we mentioned earlier, phis can be read as either 0 or 1, so they're left clear for either type of decoding. Don't forget this!!! PHIS ARE NEVER FORBIDDEN! We can use them as 0s OR 1s.

CONVERTING PRIME IMPLICANTS TO DECIMAL MINTERMS

To date we've done quite a bit of extracting prime-implicants from minterms, but suppose we were asked to do the reverse? That is, suppose we were asked to state what minterm-numbers are covered by c'e, given that this is part of a 5-variable machine using controls a, b, c, d and e.

With our decoding cards it's an easy job! We simply stack the 4-card on top of the 1-card so they read --0-1, PLUS THE 32-CARD SET TO 0, as this is the ONLY UNUSED variable. We therefore have a combined reading of 0—0-1 at the top of our cards, after which we read off all exposed numbers. If we were interested in, say, 'bc' in a 4-variable machine, we'd discard 'a' and 'd' (that is the 8-card and the 1-card) BUT RETAIN THE TWO HIGHEST CARDS SET TO 0. That is, the 32-card and the 16-card, after setting the 4-card and the 2-card to 11, to give a reading of 00-11-, and again read off all exposed decimal numbers.

Manually, it's not much more difficult if only a small number of variables is involved. In the case of our first example, c'e, we'd begin by writing --0-1 to represent c'e in a 5-variable setting, and imagine that each '-' is a '0'. This gives us our first minterm-number, 1. Now, WORKING FROM THE RIGHT, we'll convert these '-'s, one at a time, to '1's, adding the value of that particular bit-position to all decimal numbers obtained so far.

Thus our first '-' is in bit-position 2, so we add 2 to our 1 to give us a sequence of 1, 3 so far. Next we'll convert bit-position 8, which means adding 8 to our sequence, to give us 1, 3, 9 and 11 up to this point. Finally, we have to add 16 to each of these, to give a resultant minterm coverage of 1, 3, 9, 11, 17, 19, 25 and 27.

If you work from the right, as we did here, you'll find that the minterms will be developed in strict numerical order.

For the moment we'll take a break from sequential circuits, just to let it all sink in, but we'll come back to them a little further down the road. On the next leg of our journey we'll examine certain special types of combinational networks, and learn some rather strange and wonderful ways of achieving very sophisticated relay networks, many of which would be virtually IMPOSSIBLE to develop by any other means.

In lieu of a test, I'll give you a chance to go back over some of your earlier manual decodings and try them again with our decoding-cards. Maybe some of you will even want to extend them to, say, eight variables, but keep in mind that each time you add a variable, one of the central square's dimensions will double, so that for eight variables (and keeping the centre as a square) you'll have cards of dimension 4" x 5". You'll also find that extra layers of acetate begin to act like a mirror, so you'll have difficulty seeing through to the matrix below. In which case your best bet would be to begin with opaque cards and cut out the clear portions of the central square. In addition, you'll also have to cut pieces off the top and bottom edges of the cards so the 0s and 1s now stick up like little "ears" or index-tabs. You may find that you'll also have to go for larger than 1/4 x 1/4 windows in order to accommodate the much bigger minterm-numbers you'll have to write on the matrix-card.

Of course, it'd be a lot nicer if one of you wrote a program to handle this! Don't forget that when I was heavily engaged in all this stuff (going back to the late 50s and early 60s), no computer service was available at all, which is why I worked on developing these cards. But they sure paid off, and enabled me to design some very complex control systems in much less time than by my former manual method, AND MORE ACCURATELY TOO!

... End of Mile 11

+++

Pascal and Modula-2
A Tutorial

By: Robert D. Reimiller
Certified Software Corp
616 Camino Caballo
Nipomo, CA 93444
805 929-1395

This month we start our discussion of Modula-2, by way of comparison with OmegaSoft Pascal and C.

When it comes to structured languages, the two most important seem to be Pascal and C. Pascal came out about 1970 and was originally intended as a tool to teach students structured programming. It has since become a very popular language all over the world, with many different varieties to handle different applications. The C language saw the light of day around 1977 and was primarily intended to be a fairly low level language for operating systems programming. It too has expanded it's applications into many areas of computer science.

Both languages are well suited to projects requiring one or a few programmers. Large projects that require dozens of programmers sometimes fall prey to a problem of coordination between programmers using either language. This is aggravated by the lack of inter-module reference checking during compilation. Both C and Pascal allow references between modules, both procedures and variables can be declared in one module, and referenced in others. The problem lies in the lack of type checking between modules. In OmegaSoft Pascal, this type checking can be done in the debugger, but can only check those modules that are compiled with debugging information enabled. Some C packages include a separate "lint" program that does this same function, but this is lacking from all too many C packages.

In 1980 the creator of Pascal, Niklaus Wirth, published a specification for a new language, called Modula-2, that deals effectively with this problem. Modula-2 borrows much of it's syntax from Pascal, but adds the concept of "Definition" and "Implementation" modules. The definition module declares external definitions for the module, the Implementation module has the actual code to implement the procedures. With this method, the "Definition" module is provided to the other members of the programming group, and they work with this definition module. The implementation part of the module is known only to the programmer actually responsible for the module, and is free to make improvements without affecting those who reference his module.

The definition module has other advantages. First, it is used by the compiler when a programmer needs to reference procedures, types, or variables in the module. The programmer does not need to declare parameters, constant values, or anything other than the name of the item he needs to use, the compiler derives this information from the definition module. This can eliminate many potential errors in parameter passing for instance. Second, if properly commented, it provides system-wide documentation for the interface between modules. Third, it allows for so called "opaque" type declarations. Opaque types can be used so that other programmers do not know the structure of the type, they can only use the procedures that operate on that type. This can eliminate the costly problem of having everyone know the structure of a complex data type, having some programmers operate on individual fields of the structure, and having the programmer who has responsibility for that type change it's structure, creating difficult to find errors. Opaque types avoid this problem by excluding other programmers from doing anything other than what is intended to be done with the type.

In a sense, Modula-2 can be considered to be a replacement for both Pascal and C. Since OmegaSoft Pascal is an extended version of standard Pascal, and we wouldn't put out a Modula-2 compiler that was less capable than what we already have, OmegaSoft Modula-2K will be an extended version of Modula-2. Wirth's Modula-2 still comes from a University environment as does standard Pascal and in some areas does not take into account that many Microprocessor programmers are in fact, Electronic Engineers, Chemical Engineers, or anything but computer science trained, and have a different perspective of programming problems.

Note that OmegaSoft Modula-2K is not yet available. When will it be available? sometime in 1988 (that's the best I can do for an estimate right now). All references to Pascal are assumed to be OmegaSoft 68020 Pascal, unless it is noted to be standard Pascal. All references to Modula-2K are for OmegaSoft Modula-2K, Wirth's original Modula-2 is simply referred to as Modula-2. All references to C are for C as defined by Kernighan and Ritchie.

There are some basics that most languages have in common, these are :

1) Data types and constants
2) Variable allocation
3) Operators
4) Built-in procedures
5) Statements
6) Inter-module references

Lets start at the top of the list, data types. Since OmegaSoft products are targeted at real-time applications, it is important to make sure that data types can be efficiently implemented on the Microprocessor family being used (in this case the 68000 series).

One of the most basic data types is simple on/off data. Since this type of data only has two states, it can be represented in 1 bit. The 68000 doesn't allow memory accesses of one bit, so it is commonly implemented as the least significant bit of a byte, therefore it has two possible states :

```
Bit # 7 6 5 4 3 2 1 0      Bit # 7 6 5 4 3 2 1 0

ON    0 0 0 0 0 0 0 1      OFF   0 0 0 0 0 0 0 0
```

In Pascal and Modula-2K this data type is called boolean. C has no such data type, and suffers accordingly. Boolean types have two states, called true and false and are commonly used as flags, or as the result of comparisons.

The next simple data type normally encountered is used to handle ASCII characters and is called "char" in Pascal, Modula-2K, and C. It is stored in one byte of storage on the 68000 series. The ASCII character set only defines 128 values, the other 128 values are often used to handle extended character sets, or may be invalid depending on the operating system environment.

The most used group of types are called "integers" and are signed values using two's complement arithmetic. The 68000 series supports three fixed point data sizes, and so there are normally three variations :

	Pascal	Modula-2K	C
8 bit	shortinteger	shortint	not supported
16 bit	integer	integer	short and/or int
32 bit	longinteger	longint	int and/or long

Integers are commonly used for counters, numeric values, and many other uses. Note that C compilers tend to be split on what size an integer is, some of them are 16 bit, others are 32 bit. Those that use 32 bit will tend to be slower than those that use 16 bit, but possibly more compatible with UNIX C programs.

Standard integers are probably the most commonly used and are the most efficient size for the 68000 series. This is especially true on the 68000 with it's 16 bit data bus and more limited ALU, almost any instruction dealing with data will take longer if it is 32 bit than 16 bit. On the 68020 there is less of a speed difference (assuming 32 bit data is aligned on longword boundaries in memory), but even so, 32 bit operations tend to take longer than 16 bit.

Short integers (8 bit) are most likely to be used to correctly interface with hardware, such as an 8 bit signed A/D convertor. This allows it to be accessed correctly as 8 bits, but still allows operations on it to be mixed with larger signed data sizes. Another use for 8 bit integers are if you had a large array of integer values that are within the range of 8 bits, using shortintegers would take half of the space in memory to store the array. 8 Bit operations run at the same speed as 16 bit operations, except if you are tempted to use 8 bit integers as array indexes. This is because the 68000 series does not have an 8 bit index mode, only 16 and 32 bit. In this situation it has to extend the 8 bit value to 16 bits, thereby generating extra code and wasting time.

Long integers (32 bit) are used where the range of a 16 bit is exceeded. 32 bit operations usually run slower than 16 bit operations, so only use this type when necessary. The Pascal and Modula-2K compilers allocate global longintegers on longword boundaries when compiling for a 68020, so that they can be accessed in one memory reference.

Sometimes you don't want to be using signed arithmetic, for instance, dealing with an unsigned A/D convertor, or when accessing memory. In this case, there is an unsigned version of the integers :

	Pascal	Modula-2K	C
8 bit	shorthex	shortcard	not supported
16 bit	hex	cardinal	unsigned short and/or int
32 bit	longhex	longcard	unsigned int and/or long

Another reason to use unsigned values, is that in Pascal unsigned values are printed in hexadecimal format, while integers are printed in decimal format.

Although not always needed in real-time applications, floating point values are very useful to have available, especially in scientific programming.

	Pascal	Modula-2K	C (typical)
32 bit	real	real	float
64 bit	longreal	longreal	double

Both Pascal and Modula-2K use the IEEE format and support the 68881 and 68882 coprocessors, many C compilers will also.

Another data type that is useful, but not completely necessary is the enumerated type found in Pascal and Modula-2K, and found as an extension in some C compilers. This data type allows you to specify the valid names for a value, such as :

```
colors = (red, blue, green) ;
```

In essence, what you have is similar to having three shorthex constants, being defined as red = 0, blue = 1, and green = 2. These values are stored in one byte, so have a maximum of 256 possible values. In Pascal and Modula-2K the starting value can be declared using something like :

```
extendedascii = (:127, square, circle, upline,
                        acrossline . . . .
```

The value after the ":" specifies the last value used, so square will have a value of 128.

In Pascal and Modula-2K you can define a subrange of many of the non floating point data types. These are normally used for array declarations and such, for instance :

```
slotvalue = array [-10 .. 10] of real ;
```

-10 .. 10 is a subrange of integers that lets the compiler know that the valid range for indexing this array is from -10 to 10 inclusive, and tells it that this types needs room for 21 real elements. Modula-2K has a slightly different syntax for subranges. In the above declaration, the brackets are part of the subrange definition, and not the array definition. C does not have this facility, so that all arrays start at zero and all references to such an array in this example would have to have the value 10 added before doing the array access.

Beyond the simple types, we get into the structured types. The most basic structured type is the array, and is available in all three languages. In Pascal arrays are declared using :

```
ARRAY [ <index-type> ] OF <type>
```

In Modula-2K :

```
ARRAY <index-type> OF <type>
```

In C :

```
<type> <variable-name> [ <highest-index> ]
```

In all three languages arrays are accessed the same way :

```
<variable-name> [ <index-value> ]
```

Arrays store multiples of the same data type.

A variation of the array is the string. The string is normally used to store a sequence of ASCII characters, but in Pascal can be used for other data as well, regardless of it's value. In Pascal a string is defined as :

```
string              - 80 characters max
string [<limit>]    - specified maximum size
maximum)
```

In addition, the string can be indexed by a character or integer value from 0 up to the maximum declared limit. Element zero of the string is the number of characters that are currently valid in a string, for instance :

```
var
  s : string[10] ;
begin
  s := 'hello' ;

s[0] = 5 (the number of valid characters)
s[1] = 'h'   s[2] = 'e'   .s[3] = 'l'
s[4] = 'l'   s[5] = 'o'
```

elements 6 through 10 contain garbage data.

In Modula-2K and C, strings are simply arrays, so for a 10 character string :

```
Modula-2K :  s : array [0 .. 9] of char
C :          char s[10]
```

In both languages the end of the string is marked with a zero byte, so for instance, in Modula-2K :

```
s := 'test' ;

s[0] = 't'   s[1] = 'e'   s[2] = 's'
s[3] = 't'   s[4] = 0
```

elements 5 through 9 contain garbage data.

Which one is better? actually neither. The pascal method allows any value to be represented in the string since it does not use a terminator. For instance, this is used in the assembler and linker to hold relocatable object records. It is also very easy to determine the length of a Pascal string. On the other hand, the Pascal method limits the size of the string to 255 characters maximum.

The Modula-2K and C methods allow for a string of any length, up to the limit of memory. On the other hand, you cannot use the value zero in your string, which limits it's uses, and to determine the length of a string you have to scan the entire string until you find a zero. One advantage the Modula-2K and C methods do share is that many operating system use the same format for file names, so there is no conversion needed as is sometimes required with Pascal strings.

Another more sophisticated data type is the structure, called a "record" in Pascal and Modula-2K, and a "struct" in C. This data type allows you to put together other data types into a single entity :

```
PASCAL AND MODULA-2K                    C

customer = record                  struct customer {
             name : string ;            char name[80] ;
             age : integer ;            int age ;
             balance : longreal         double balance ;
           end ;                      } ;
```

For all three languages, an individual element of a structure is accessed by using the structure name, a period, and then the field name, such as :

```
customer.age := 31            customer.age = 31
```

The SET data type is supported by both Pascal and Modula-2K, but is not available in C. Sets are a mathematical abstraction for a collection of objects. For instance, consider the ability to generate 8 colors (black, white, the three primary colors, and 3 mixtures of the primary colors). In Pascal you would define the three primary colors as :

```
type
  colors = (red, green, blue) ;
```

And the collection of these colors as :

```
var
  primaries : set of colors ;
```

The type primaries then can contain any combination of colors, the empty set (nothing in it) in this case would represent black, all colors in the set would be white (assuming we are talking about mixing light here, and not pigments).

```
primaries := [] ; (* black *)
primaries := [red, green, blue] ; (* white *)
primaries := [red] ; (* red *)
primaries := [green, blue] ; (* yellow *)
```

The syntax and capabilities in Pascal and Modula-2K are different, even though the concept is the same. The above examples were in Pascal. The declarations are the same in both, but the syntax for a set constant is different, in Modula-2K white would be :

```
primaries := {red, green, blue} ; (* white *)
```

Modula-2K uses curly braces instead of brackets for set constants. Pascal uses a dynamic length set structure very similar to a string where the first byte is the current length of the set. The most significant bit of the next byte is the highest set element, down to the least significant bit of the last byte, which is element zero. The internal representation in Pascal for yellow is :

```
 SET LENGTH (1)         BITMAP
7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0

0 0 0 0 0 0 0 1    0 0 0 0 0 1 1 0

B G R <- COLORS
```

Since in the runtime routines used, the maximum length of data bytes for this type of storage is 252, this gives a set range of 0 through 2015.

The Modula-2 specification is a bit more specific about how sets are to be represented, at least for those that will fit in a machine word, longer sets are considered optional. In Modula-2K sets are stored in one byte, two bytes, four bytes, or a multiples of 4 bytes. A set that fits into one byte is any that has a maximum element of 7 or less. Likewise, a two byte set goes up to an element of 15, and a four byte set goes up to 31. The four byte set is also a predefined type called a "BITSET". Sets who have their maximum element greater than 31 are stored as multiple of 4 bytes, for whatever size they need to be, within in limits of available memory.

This makes sets useful when dealing with I/O ports. For instance if you have an 8 bit I/O port that has 8 lines, the first four (bits 0-3) are input switches, the other four (bits 4-7) are lights. This port could be defined as :

```
type
  switch = {left, right, up, down} ;
  lights = {down: error, proceed, standby, destruct}
;
var
  control [$F3400] : set of left .. destruct ;
```

The [$F3400] tells the compiler that this one byte set variable is to be located at hexadecimal location F3400. One could check to see if the up switch is on by using :

```
 if up in control
    then
       <perform operation>
```

You can turn on the standby light (without affecting the other lamps) by using :

```
   incl {control, standby}
or  control := control + {standby}
```

The more conventional method would be :

```
const
  up = 4 ;
  standby = $40 ;
var
  control [$F3400] : shorthex ;
 begin
  if control and 4 <> 0
    then
      <perform operation> ;
  control := control or $40
```

A similar capability (but not using set syntax) is available in C and is called bitfields. It is not applicable for structures larger than 4 bytes, and bitfields are implemented in only a few compilers.

Pascal defines I/O devices in the language itself, and so has a device type, which is a more general version of the standard Pascal file type. Modula-2K and C provide I/O facilities using external procedures, so no real comparison can be made here, we will cover this in a later month.

The last data type is the pointer type, which is very important in most real-time and systems programming. Pointers are essentially the address of something, and so by definition on the 68000 series they are 4 byte quantities. A pointer to an integer is declared something like :

```
PASCAL             MODULA-2K                  C

intpt : ^integer  intpt : pointer to integer  int *intpt
```

Assuming we have put a value into a pointer, we can access the data it points to by "dereferencing" it, for example to assign the value of 5 to where the pointer points to :

```
    PASCAL AND MODULA-2K                C

          intpt^ := 5                *intpt = 5
```

The main use of pointers is in dynamic data structures, where a block of memory is obtained from the environment and used to store information. Since the compiler cannot know where this memory will be, a pointer is used that has it's value determined at run-time.

Pointers are often used in C to access I/O ports since C lacks a simple way to determine at what address a variable will located, in this case the pointer is loaded with the location of the I/O port. Most compilers will accept something called a "caste" that does allow access to a specific location without using a pointer, but the syntax is a bit murky and will be discussed in the section of variable allocation, which will be in a later article.

Now let's talk about constants. Boolean constants are TRUE and FALSE for Pascal and Modula-2K, C doesn't have a boolean data type. Character constants have a number of forms in each language, depending on whether or not the character value is printable and what base you to use.

In Pascal printable character constants can be specified by putting the character in between single quotes, such as 'X'. If you need to specify a single quote as the character, then use two single quotes next to each other, which will be treated as a single character and not as a delimiter, such as ''''. Numerical values can be specified as well, for instance, each of these represents decimal 15 :

DECIMAL	HEXADECIMAL	BINARY
#15	#$F	#%00001111

In Modula-2K printable character constants can be specified by putting the character in between either single or double quotes (but not a mixture of the two), such as 'X' or "X". Double quotes are normally only needed if the character you want to represent is a single quote, the dual single quote trick doesn't work for Modula-2. Numerical values can be specified as well, for instance, each of these represent decimal 15 :

DECIMAL	HEXADECIMAL	BINARY	OCTAL
\15	\$F	\%00001111	17C

. In C printable character constants can be specified by putting the character in between single quotes, such as 'X'. A single quote is represented by \' . Numerical values can be specified, but only in octal (who uses octal anymore?), decimal 15 would be :

OCTAL
\17

Integer constants are normally written as a series of decimal digits. Decimal constants (without an exponent or decimal point) are considered to be integers unless they are larger than integer size, in which case it is longinteger. In some rare cases, you might want to explicitly specify that an integer is short or long, in this case you can follow the number with the letter "L" to specify long. In Pascal and Modula-2K, a short integer can be specified by following the number with the letter "S", C apparently has no method of specifying a short integer constant. Unsigned constants can also be specified using a variety of forms, the following examples use the decimal value 15 again :

	PASCAL	MODULA-2K	C
short integer	15s	15s	not supported
integer	15	15	15
long integer	15L'	15L	15L
unsigned (hexadecimal)	$F	$F	0XF
unsigned (binary)	%00001111	%00001111	not supported
unsigned (octal)	not supported	17B	017

Floating point constants are pretty much standardized between languages, and are available in two formats, with and without an exponent, such as :

```
    4.3     0.3e-5     4037E205
```

In Pascal and Modula-2K these would be considered as real constants, in C they are double constants (there are no float constants in C). In order to get a longreal constant in Pascal and Modula-2K we need to do one of two things, either follow the number with the letter "L" or use the letter "D" for the exponent part instead of "E".

Array and Record constants are handled by all three languages, but their syntax has its roots in variable declaration and allocation, so let's leave that one until later.

String constants are very similar to character constants, but include more than one character (a string constant with only one character is generally considered to be compatible with characters). Pascal again uses the single quotes, and to represent a single quote you use two of them together. Modula-2K uses single or double quotes, at the programmer's whim. C uses double quotes. How do we add non-printable characters to a string? We use "escape" characters to do it. For sake of example lets assume we want to have the string "test", followed by decimal 15, followed by" done".

PASCAL	MODULA-2K	C
'test'#15' done'	'test'\15' done'	"test\17 done"

Set constants were discussed in the section on sets.

There is really no such thing as a pointer constant, you just use a signed or unsigned constant.

Next time we look at how variables are allocated and accessed.

OmegaSoft is a registered trademark of Certified Software Corporation, UNIX is a trademark of Bell Laboratories.

+++

Editors Note: This is part 1 of a 2 part series. Part 2 was published ahead of part 1. Please refer to the June issue for part 2. My apologies: Larry Williams

Bit-Bucket

By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

Dear Don,

The recent request for articles on the ATARI 68000 machines and the interesting article by Dale Randall, prompted me to expand on his list of operating systems by describing the ATARI versions of OS/68K and IDRIS.

I too started out with a SWTP 6800 in May of 1976, though with only 2K of memory! It is now a 2 MHZ 6809 with about 340K (including almost all FLEX utilities in EPROM). The unit is still used several times per week, though I no longer write any new software for FLEX. Its primary use is to serve the ATARI for testing programs and passing files between the many ATARI operating systems. For many years I claimed I would never allow an "appliance" computer in my house. My aversion to "closed box" computers softened when I realized the only way I could justify a 68000 machine was to buy an ATARI ST or COMMODORE AMIGA. I won't go into the "68000 WARS", suffice it to say our group of Motorola fans bought ATARI machines.

Dale has already discussed GEM, MS-DOS, CP/M, and MACINTOSH so I will move on to what Don Williams calls "serious software." I purchased OS/68K from TLM Systems back in September 1986. It cost more than the computer and was a long time coming, but well worth the wait. I found Microware's part (the operating system) to be very smooth. TLM's part (the ATARI specific programs) was rather sloppy. I soon developed a rapport with TLM as I sent in pages of bug reports. Unfortunately TLM went belly up before all of the bugs were fixed. Microware took over the ATARI version of OS/68K and eventually updated me to Version 2 and solved most of the ATARI specific problems.

OS/68K runs well on the ATARI, though it doesn't take advantage of the ATARI capabilities. Like most UNIX style operating systems it was intended for a separate serial terminal. The promised windows never arrived (the COCO III is still the only windowed OS/9 I'm aware of). The system is fast, much faster and more responsive than UNIX. The 68000 running at 8 MHZ with 1024K of RAM makes an effective engine for OS/68K. For those unfamiliar with Version 2, the old load and link method of keeping often used programs in memory has been supplemented with the "Sticky or Ghost BIT." It's sort of a "command cacheing" system. As you use utilities they are kept in memory unless a memory hungry program needs the space. Thus before long I have the complete compiler, editor, and several utilities in RAM yet I still have room for a 256K RAM-DISK.

OS/68K has many other features I like, such as recall of the command line (control A). This is enhanced by allowing trailing options. For example if you enter:

```
copy /D0/source/c/dc.c /h0/source/c/dc.c
```

and you are greeted with an error saying the file exists, you need only type Control A followed by -r. On most UNIX systems you would have to type the line over, inserting the option between the command and the arguments. I enjoy having the ability to do a sysgen, all other systems I've worked with were shipped "Like it AS IS or leave it." I have added drivers and utilities to the Os9Boot file to customize it to my needs. Microware expects you to spend more than the cost of the operating system for information on adding drivers; I spent some educational nights with a reverse assembler and was able to add several new drivers.

On the negative side, OS/68K is not UNIX. I work with UNIX on a variety of systems, and it is AWKWARD to go back and forth. I have added several UNIX-like commands to Microware's command set, but still lack several more that are too UNIX dependent to port over. OS/68K procedure files are light years behind UNIX script files, they lack parameter passing, conditional tests, and looping. Slowly Microware is heading that way, I've noticed several UNIX-like enhancements in Version 2.

I also suspect Microware really didn't want to take up ATARI support when TLM went down the tubes. Their response to most questions involving improving the ATARI port is "Perhaps some day." They even dropped several features TLM had developed, such as control over the display and keyboard.

On to IDRIS! For those unfamiliar with Whitesmith's IDRIS system, it is a UNIX compatible system running on several Motorola 680XX and DEC PDP/11 machines. It meets the IEEE POSIX and ANSI standards for UNIX compatible operating systems and C compilers. I received it February 1988, and have enjoyed working with it. IDRIS is close to UNIX V6, but some utilities are of later vintage. There was nothing in the advertising to indicate it would be three generations behind the current release of AT&T UNIX; except perhaps the mention that ATARI lacks memory management. This has not been too much of a limitation, just an inconvenience. I have already replaced a few utilities with newer versions, soon I hope to replace the shell. IDRIS is so close to UNIX it is possible to port in programs (shell, find, ls) that deal with the disk structure at fairly low level.

UNIX systems are SLOW compared with operating systems aimed at smaller computers. The disk structure slightly resembles the linked list of FLEX except the links are kept in tables called INODES (pronounced "eye-nodes"). File writes are done one 512 byte block at a time, adding block numbers to the list as you proceed. Traditional UNIX programmers are resource frugal to the extreme; I've seen a program make thousands of calls to the operating system to allocate 2 bytes at a time rather than a few calls for larger chunks of memory. IDRIS doesn't keep finished programs around like OS/68K, nor does it share one copy of a program. If while editing one file I call the editor with another file I will have two copies of the editor in memory. It does have disk sector cache, however.

The ATARI port of IDRIS was done by Computer Tools International who deserve credit for a job well done. Unlike TLM my bug list is quite short (can't get the Memdisk to work, a few manual typos). Unlike Microware they seem to like the ATARI: included are utilities to deal with the display, keyboard, mouse, and MIDI ports. Both OS/68K and IDRIS support the parallel and serial ports, as well as the floppies and one hard drive. IDRIS has a program to catalog and retrieve GEM files.

What article comparing operating systems would be complete without some speed tests? The following time trials were done on an ATARI hard disk with no effort to "tune" the operating systems. Since I don't have the IDRIS Memdisk working, I tested OS/68K with and without anything loaded in RAM. The program was found on the back cover of 68 MICRO JOURNAL. Examining the source code generated revealed the OS/68K compiler generated a three instruction loop whereas the IDRIS compiler generated a four instruction loop. In fairness to the 6809, I should mention that for 60000 loops (register short) FLEX beat the 68000. All this should be taken with a grain of salt since it only tests a tiny facet of the compiler.

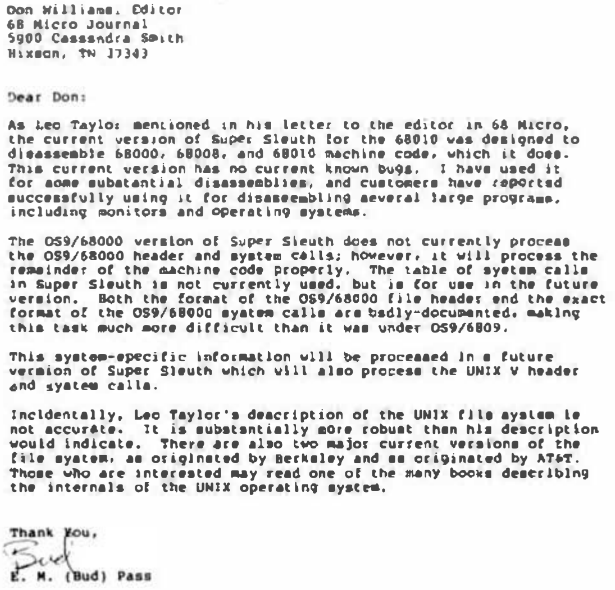
```
main()
{
    /* int i; */
    register long i;
    for(i=0;i<999999;++i);
}
```

	FLEX	GEM	OS/68K	IDRIS
COMPILE DISK	35.	12.	27.8	49.
COMPILE RAM	24.	7.	10.	
RUN INTEGER	76.	9.	6.8	9.
RUN REGISTER		6.	4.2	6.
COPY 100K FILE	45.	5.	15.	16.

Finally I'd like to comment on the "serious software" nature of OS/68K and IDRIS vs the "game-toy" nature of GEM mentioned by Don Williams. So far I have about 500 application programs for GEM, 2 for OS/68K and 0 for IDRIS. That is not to say UNIX style applications are not available, just that the few I'm interested in are prohibitively expensive. I spent $200 for an OS/68K dissassembler (SLEUTH) and found it wasn't OS/68K compatible (**Editor's note: *see letter below from Dr. Bud Pass.**) Oh it functions, but it doesn't recognize OS/68K binary files, doesn't recognize OS/68K headers, doesn't recognize OS/68K system calls, occasionally crashes by reading address zero, and even generates some mnemonics that the OS/68K assembler doesn't handle. It seems Bud Pass never finished the product since there is a large array of OS/68K system calls in the source he never used! Over on the "game-toy" operating system (GEM) I have dissassemblers that run circles around SLEUTH. Though I dearly love the popular STYLO word processor (I bought the source years ago from Sonex Systems), there are GEM word processors that have modern features that STYLO lacks. A few examples are: Built in Dictionary which suggests correct spelling, and inserts the corrected word. Thesaurus to check your choice of words as you write. Italics, bold, and even type faces on screen. Cut and paste between documents in four windows. Surely anyone who has used a mouse machine like the Apple Macintosh knows you don't have to have OS/68K to do "serious" computing. Rather, systems that use serial terminals instead of bit mapped displays are limited for serious word processing, graphics, CADD, and desktop publishing.

In conclusion I'll say OS/68K and IDRIS are keeping me busy doing what I like best: writing and porting C programs. FLEX users know I have always preferred writing utilities to do things rather than actually doing them! If running applications was my forte, I'd find more programs to run with GEM, PC-DITTO, or MAGIC-SAC.

Leo Taylor
109 Twin Brook Road
Hamden, Conn. 06514
(203) 387-9658



Don Williams, Editor
68 Micro Journal
5900 Cassandra Smith
Hixson, TN 37343

Dear Don:

As Leo Taylor mentioned in his letter to the editor in 68 Micro, the current version of Super Sleuth for the 68010 was designed to disassemble 68000, 68008, and 68010 machine code, which it does. This current version has no current known bugs. I have used it for some substantial disassemblies, and customers have reported successfully using it for disassembling several large programs, including monitors and operating systems.

The OS9/68000 version of Super Sleuth does not currently process the OS9/68000 header and system calls; however, it will process the remainder of the machine code properly. The table of system calls in Super Sleuth is not currently used, but is for use in the future version. Both the format of the OS9/68000 file header and the exact format of the OS9/68000 system calls are badly-documented, making this task much more difficult than it was under OS9/6809.

This system-specific information will be processed in a future version of Super Sleuth which will also process the UNIX V header and system calls.

Incidentally, Leo Taylor's description of the UNIX file system is not accurate. It is substantially more robust than his description would indicate. There are also two major current versions of the file system, as originated by Berkeley and as originated by AT&T. Those who are interested may read one of the many books describing the internals of the UNIX operating system.

Thank You,
Bud
E. M. (Bud) Pass

MICRONICS
RESEARCH CORP.
Microcomputers - Hardware and Software
GIMIX® Sales, Service and Support

33383 LYNN AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA,
CANADA, V2S 1E2

Dear Don,

This ia by way of a continuation of my last letter, where I ran out of room before I could get down to my expansion of XBASIC XPLANATIONS. So here goes with what I think will be a useful discussion of the logic functions OR and AND. As usual, we'll begin with the simple and work up to the more complex. So let's look at a fairly common occurrence in BASIC programs.

```
   100  IF X% < 3 GOTO 120
   110  Y% = 9
   120  rest of program
```

Nothing to do with logic functions yet, but observe that the program will fall through to line-110 only if X% >= 3, and then carry on to line-120. Therefore we could re-write this segment as

```
   100  IF X% > 2 THEN Y% = 9
   120  rest of program
```

provided there are no calla elsewhere to line-110. No, the "2" in line-100 isn't a typo! Because X% (an integer) can only take on the values ... 1, 2, 3, 4 ..., then "> 2" is obviously the same as ">= 3". Now, how about

```
   100  IF X% < 3 GOTO 1000
   110  IF Y% = 9 GOTO 1000
   120  rest of program
```

Always provided there are no other calla to line-110, we note that we get directed to line-1000 under two conditions (i) IF X% < 3, or, failing that, (ii) IF Y% = 9. So we can re-write this baby as

```
   100  IF X% < 3 OR Y% = 9 GOTO 1000
   120  rest of program
```

But suppose the program had read

```
   100  IF X% < 3 GOTO 120
   110  IF Y% = 9 GOTO 1000
   120  rest of program
```

I hope you'll all resist the temptation to shorten this one by merely swapping lines 100 and 110, and then eliminating the new 110 as being redundant. Let's study it a little further to get the "logic" of it all.

Our first observation is that the program will fall through to line-110 if X% > 2 (remember our earlier example), and then only if Y% = 9 will it shoot off to line-1000. Therefore the correct re-structuring of this would be

```
   100  IF X% > 2 AND Y% = 9 GOTO 1000
   120  rest of program
```

Got it so far? OK then, let's look at

```
   100  IF X% < 3 GOTO 120
   110  IF Y% = 9 GOTO 1000 ELSE 2000
   120  rest of program
```

After our previous example this one's easy! Just re-write it thus

```
   100  IF X% > 2 AND Y% = 9 GOTO 1000 ELSE 2000
   120  rest of program
```

Agreed? If you said YES, then you'd be wrong, wrong, wrong! This is not the same program any more! Why not?, you ask. Well, for one thing, note in our original program that having reached line 100, it's possible (if X% < 3) to go to line-120. But that ELSE makes all the difference in the world, as there's now **no way** at all for our new program to reach line 120. From line-100 it can only go to line-1000 OR ELSE to line-2000. So ... if you're tempted to re-write a line containing an IF-THEN-ELSE, my advice is to think again, and preferably resist the temptation altogether, or else be prepared for trouble.

Now for something a little different. Suppose, to keep it simple, we've written a game where, let's say, three or more players in turn key in a random letter of the alphabet (each one unseen by the others), and the player who keys in a letter which matches that keyed in by the player 2 positions back becomes the winner. Thus in the sequence QBGXMLM, the player who keyed in the final "M" would be the winner, as there is a match 2 positions back!

Our game, then, builds up a string P$ as each letter, X$, is entered, and the program would check thus, where X% is a count of letters keyed in.

```
    10  request input (X$), bump X%, and add X$ to P$
   100  IF MID$(P$,X%-2,1) = X$ GOTO 200
   110  PRINT "No match": GOTO 10 (ask next Player)
   200  PRINT "You win!": END
```

Do you see why this program would bomb? Problem is that for the first two responses, P$ wouldn't be long enough for us to look back by two positions, so "X%-2" would cause the MID$ function to fail. OK then, let's change the essential part of this program to

```
   100  IF X% < 3 GOTO 120
   110  IF MID$(P$,X%-2,1) = X$ GOTO 200
   120  PRINT "No match": etc
   200  PRINT "You win!": END
```

There, that should take care of that! Agreed this time? Take another look before you answer! If you said YES this time you'd be correct. But then, by applying our earlier principles, and noting that we fall through to line-110 only if X% > 2, we can now shorten this to

```
   100  IF X% > 2 AND MID$(P$,X%-2,1) = X$ GOTO 200
   120  etc
```

and we're still in business! Right? Although this seems intuitively correct, we'd be be creating problems with this approach, because XBASIC and several other BASICs too (including RBASIC), evaluate a **complete** logic function before making a final decision. Thus, even though X% were, let's say, only equal to 1, it would not **immediately** fail and fall through to line-120, but would carry on to evaluate the remainder of the logic function coupled to it by the AND statement. And would bomb under these conditions in its evaluation of P$.

On the other hand, the following

```
  100  IF X% > 2 THEN IF MID$(P$,X%-2,1) = X$ GOTO 200
```

would work OK, because there are now two distinct conditional IF-THENs to check, and a "fail" on the first (X% > 2) would immediately drop through to line-120. So you see that AND is not **always** equivalent to

THEN IF (certain publications notwithstanding), though it would be in the following instance

```
   100  IF X% < 3 AND Y% = 9 GOTO 200
and
   100  IF X% < 3 THEN IF Y% = 9 GOTO 200
```

The difference, of course, is that the second conditional (Y% = 9) is independent of the first, unlike our P$ example, where it's essential that the "X% - 2" in the second conditional evaluate to "1" or more.

A further example to round out this part of my discussion. Beginning with, let's say,

```
   100  IF X% < 3 GOTO 120
   110  IF Y% = 9 THEN Z% = 5 ELSE Z% = 17
   120  rest of program
```

the program has to arrive at line-120, one way or another, with Z% set to some value, so it should be perfectly OK to re-write this as

```
   100  IF X% > 2 AND Y% = 9 THEN Z% = 5 ELSE Z% = 17
   120  rest of program
```

Here, too, we'll arrive at line-120 with Z% set to some value, so everything's OK! Again, take a good hard look rather than just blindly agreeing with me. And again, if you did agree, you'd be wrong! What's wrong this time? Well, in our original program, it's possible for Z% to arrive at line-100 with any value, and under the condition (X% < 3) to retain this value as it gets re-directed to line 120. But, if (X% > 2) it would fall through to line-110, and change Z%'s value to either 5 or 17, depending on the second conditional clause (Y% = 9).

In our re-write, however, the original value of Z% will **always** be changed to either 5 or 17 before continuing to line-120, with absolutely **no** possibility of retaining its original line-100-value. So, here too, if we wish to retain the intent of the original, we MUST replace our AND with a THEN IF, even though the two conditionals, depending on X% and Y% respectively, are apparently independent. So this is the correct form

```
 100 IF X% > 2 THEN IF Y% = 9 THEN Z% = 5 ELSE Z% = 17
 120 rest of program
```

So be **very careful** with those IF-THEN-ELSE combinations, and **always** check against the intent of the original!!!

Some BASICs, even in our P$ example, decide (correctly) that if the first conditional is FALSE then it's pointless to check any other conditional linked to it by AND, and would fail immediately. This capability, however, takes a lot more code to implement in a BASIC, which is why I, and others, take the easy route and evaluate the whole function, no matter what!

Similarly with an OR function. The moment one of the OR conditions is TRUE, it would logically be pointless to check any other conditional linked by OR, but again ... more code.

I'll have more to say on this subject later, but for now, enough is enough!

Sincerely,

Bob

R. Jones
President

Don Williams,
68 Micro Journal,
5900 Cassandra Smith Road,
Hixson, TN 37343

Noel M.Moss
14 N. Kingshighway Blvd.
St. Louis, MO 63108

Gentlemen:

I recently discovered a resource that your more technically oriented subscribers may be interested in. Motorola runs a "Freeware" bulletin board system that has downline loadable cross assemblers, cross compilers and utilities. Theese tools are intended for use on PC's, MACs and a few other machines.

The cross assemblers include the following target machines: 6800/02, 6801, 6804, 6805, 6809and 68HC11. There are also a few small C-compilers, a Kermit utility and assorted odds and ends. The tools seem to work within certain limitations such as little or no macro support, etc., but they are FREE.

I had occasion to use the 6800 PC resident cross assembler for a recent project to update some ancient code and found that the assembler reported "Unrecognizable Mnemonic" for the ABA instruction. I have left a query on Motorola's message system but have not had a response yet. The simple fix to this problem is to code:

```
              ABA     EQU     $1B
then insert:  FCB     ABA
```

whereever ABA is needed. This approach also allows one to implement 6801 unique mnemonics on a 6800 assembler.

The telephone number for the freeware system is:
512-440-3733

The system supports 300, 1200 and 2400 BPS. When it answers, simply type one or two carriage returns for autobaud. It seems to be up 24 hours per day but I have run into a few instances where the modem answered but the system wasn't home.

Sincerely,
Noel M. Moss

Dear Sirs:

In past issues you had letters from readers printed. I've not seen them in recent issues. If possible could you ask this question? Does anyone still use the 6800 CPU? And if so would he please write me. I'm still using this CPU.

Also I would like to know if some of the older companies still have or support this CPU. Companies like CSS, TCS, Microware, etc. If they do, can you give me there present address?

Sincerely,
John J. Fiorino
518 85th Street
Brooklyn, NY 11209

P68000 µLAB™

University Research and Development Associates, Inc. (URDA, Inc.) has just announced The P68020 µLAB™ microprocessor development system, the fourth in the URDA µLAB™ series. The P68020 µLAB™ is a Motorola 68020 32 bit microprocessor with support chips, optional floating point coprocessor (68881), 32 K SRAM, 8 K EPROM, Monitor, 32 bit program operation from the SRAM and User Manual.

Packaging is a single board expansion for the P68000 µLAB™ through two 50 conductor ribbon cables. The P68020 µLAB™ uses the keypad, display, power supply, etc., of the P68000 µLAB™ which is required for operation and sold separately by URDA, Inc.

The P68020 µLAB™ is a state of the art 32 bit microprocessor system providing individual hands on experience at an economical price. Several options are available. A complete operational system including both the 68020 and 68881 (floating point coprocessor) chips as well as a serial port and a 68020/68881 cross assembler hosted on a PC costs only $899.50 (Separately Purchased Price $924.45)

To update a P68000 µLAB™ that you now own, deduct $197.50. If you don't need a floating point coprocessor, deduct $145. Thus, to upgrade your current P68000 µLAB™ to the 32 bit Motorola 68020 with 32 K SRAM, the cost is only $372.50.

Traditional development systems require many times more cash expenditure, laboratory space, laboratory monitoring and scheduled access. The µLAB™ overcomes all of these limitations. The µLAB™ is engineered to be low cost providing hands on experience for engineers, technicians and students.

The User Manual contains a complete description of the system operation including explanation of all key functions, memory map, software utilities, and programming examples including Sound (Tone) Generation, and Visual Display examples.

The user can conveniently carry the Notebook Computer™ to the home, dormitory, or other work or study location to experiment with the microprocessor at his/her convenience providing laboratory type capability, or it can be used in conjunction with lecture type courses without the additional laboratory expense and inconvenience.

The P68020 µLAB™ and previously announced µLAB™ microprocessor development systems provide for inexpensive simultaneous access to a number of processors heretofore available only on a "one at a time" basis from expensive multiple target microprocessor development systems.

Optional accessories for the µLAB™ include:

(1) P68681 µLAB™ serial I/O to the IBM-PC® and Macintosh® at $59.50

(2) P68020ASM Cross Assembler to run on an IBM-PC® or suitable clone, at $149.95

(3) PADC-DAC-8 µLAB™ 8 channel, 8 bit analog to digital/digital to analog converter at $197.50

(4) PWWEB-2 Wire Wrap expansion with 6 documented experiments and wire wrap tools at $50.00

(5) PBURN µLAB™ EPROM Burner for 2716, 2732, 2764, 27128 and 27256 EPROMs

A Summary of the P68020 µLAB™ features:

- All sockets, wiring, and support chips for the Motorola 68020 32 bit microprocessor (the Motorola 68020 chip must be ordered as a separate unit)
- 4 K bytes Monitor and Utility EPROM
- 4 K bytes User expansion EPROM
- 32 K bytes of static RAM
- 32 bit bus out of SRAM
- Detailed User's Manual including schematics, programming instructions, and operating system listing

FOR PRICES AND ORDEIRING, CALL URDA, Inc.:

University Research and Development Associates, Inc.
4516 Henry Street, Suite #407
Pittsburgh, PA, 15213
1-800-338-0517 or 1-412-683-8732

WINDRUSH
Micro Systems Ltd

Winstead Laboratories Reg. Offices
North Walsham, Norfolk NR28 9SA
Tel. (0692) 404086
Telex 975548 WANDRO G
MAPCON approved consultants

Low Cost MC68020 SBC Serves as MDS or Te get

Windrush Micro Systems Limited are pleased to announce the immediate availability of low-end Ωmega workstations for the educational and OEM target system market.

The basic system incorporates a 12.5 MHz MC68020, 512K of zero wait-state static RAM, five RS-232C serial ports and a parallel printer port.

This system is supplied with Floppy and SCSI interfaces (but no drives) and OS-9/68K Professional for £1,595.00 (10 off) or OS-9/68K Industrial for £1,495 (10 off).

A single 1 mb 3.5" floppy disk drive adds £135 and a single 20 mb Hard disk adds £375.

The standard range of Ωmega I/O cards can be used with these low cost models. These include: A colour graphics adaptor which provides a 768 x 576 x 4 bits/pixel resolution. Up to sixteen colours from a pallette of 4096 colours may be displayed simultaneously. Other options include a 9-port serial expansion card, a 3 Mb ROM expansion card, 100 - 340 mb hard disks and 150 mb tape backup.

The top of the line Ωmega workstation, costing £4950 incorporates a 16 MHz MC68020 processor and MC68881 math co-processor as standard and includes five RS-232 ports, a 40 Mb Winchester hard disc with a seek time of less than 30 mS, a 1 Mb 3.5 inch floppy disk, a 150 Mb 1/4" tape streamer, 2 megabytes of zero wait-state, non-volatile Static RAM. A parallel printer port, a clock calendar and OS-9/68K professional are also included.

For further information contact Bill Dickinson at (0692) 404086

Joyce Williams
68 MICRO JOURNAL
5900 Cassandra Smith Road
Hixson, TN 37343

May 13, 1987

Dear Mrs. Williams,

The enclosures identify the various host/target models of C cross-compiler systems, Modula-2 cross-compiler systems, and macro relocating cross-assemblers that we're offering these days. As you are probably aware these cross-development products are several generations newer and significantly more 'expanded' and more capable than the resident C compilers we were selling for UniFlex, Flex, and OS9 several years back. Also, you'll notice that although we now support a very wide variety of development hosts, none of these newer and enhanced products are available for 6809-based development systems.

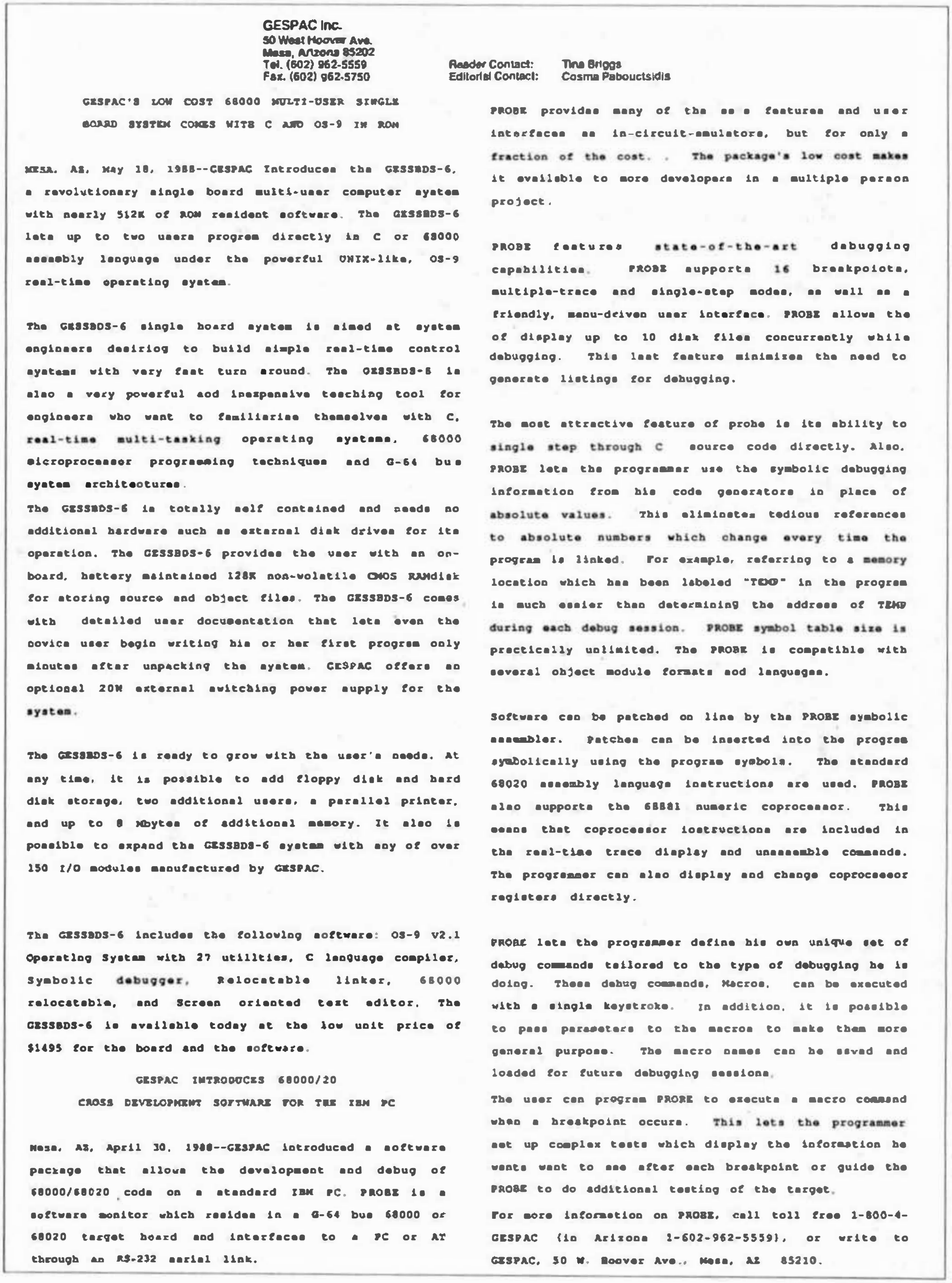
As I'd mentioned when you phoned yesterday, although our current family of products is oriented towards users who are doing their software development on PC/XT/ATs, VAXes, and various of the more popular workstations (SUN, Apollo, HP 9000, etc) we can still supply copies of our older, version 1.6 resident C compilers under UniFlex, Flex, and OS9 and the retail price for those packages is still $575. Though we haven't advertised nor promoted the availability of our 6809-based software tools for the past 3 or 4 years now, surprisingly enough we still do get an occasional order for a UniFlex or OS9 copy from time to time.

Glad to hear everything is going well at 68 Micro Journal. Please give my regards to Don.

Best regards,

John Wiatalowski

I N T R O L



GESPAC Inc.
50 West Hoover Ave.
Mesa, Arizona 85202
Tel. (602) 962-5559
Fax. (602) 962-5750

Reader Contact: Tina Briggs
Editorial Contact: Cosma Pabouctsidis

GESPAC'S LOW COST 68000 MULTI-USER SINGLE BOARD SYSTEM COMES WITH C AND OS-9 IN ROM

MESA, AZ, May 18, 1988--GESPAC Introduces the GESSBDS-6, a revolutionary single board multi-user computer system with nearly 512K of ROM resident software. The GESSBDS-6 lets up to two users program directly in C or 68000 assembly language under the powerful UNIX-like, OS-9 real-time operating system.

The GESSBDS-6 single board system is aimed at system engineers desiring to build simple real-time control systems with very fast turn around. The GESSBDS-6 is also a very powerful and inexpensive teaching tool for engineers who want to familiarise themselves with C, real-time multi-tasking operating systems, 68000 microprocessor programming techniques and G-64 bus system architectures.

The GESSBDS-6 is totally self contained and needs no additional hardware such as external disk drives for its operation. The GESSBDS-6 provides the user with an on-board, battery maintained 128K non-volatile CMOS RAMdisk for storing source and object files. The GESSBDS-6 comes with detailed user documentation that lets even the novice user begin writing his or her first program only minutes after unpacking the system. GESPAC offers an optional 20W external switching power supply for the system.

The GESSBDS-6 is ready to grow with the user's needs. At any time, it is possible to add floppy disk and hard disk storage, two additional users, a parallel printer, and up to 8 Mbytes of additional memory. It also is possible to expand the GESSBDS-6 system with any of over 150 I/O modules manufactured by GESPAC.

The GESSBDS-6 includes the following software: OS-9 v2.1 Operating System with 27 utilities, C language compiler, Symbolic debugger, Relocatable linker, 68000 relocatable, and Screen oriented text editor. The GESSBDS-6 is available today at the low unit price of $1495 for the board and the software.

GESPAC INTRODUCES 68000/20 CROSS DEVELOPMENT SOFTWARE FOR THE IBM PC

Mesa, AZ, April 30, 1988--GESPAC introduced a software package that allows the development and debug of 68000/68020 code on a standard IBM PC. PROBE is a software monitor which resides in a G-64 bus 68000 or 68020 target board and interfaces to a PC or AT through an RS-232 serial link.

PROBE provides many of the same features and user interfaces as in-circuit-emulators, but for only a fraction of the cost. . The package's low cost makes it available to more developers in a multiple person project.

PROBE features state-of-the-art debugging capabilities. PROBE supports 16 breakpoints, multiple-trace and single-step modes, as well as a friendly, menu-driven user interface. PROBE allows the of display up to 10 disk files concurrently while debugging. This last feature minimizes the need to generate listings for debugging.

The most attractive feature of probe is its ability to single step through C source code directly. Also, PROBE lets the programmer use the symbolic debugging information from his code generators in place of absolute values. This eliminates tedious references to absolute numbers which change every time the program is linked. For example, referring to a memory location which has been labeled "TEMP" in the program is much easier than determining the address of TEMP during each debug session. PROBE symbol table size is practically unlimited. The PROBE is compatible with several object module formats and languages.

Software can be patched on line by the PROBE symbolic assembler. Patches can be inserted into the program symbolically using the program symbols. The standard 68020 assembly language instructions are used. PROBE also supports the 68881 numeric coprocessor. This means that coprocessor instructions are included in the real-time trace display and unassemble commands. The programmer can also display and change coprocessor registers directly.

PROBE lets the programmer define his own unique set of debug commands tailored to the type of debugging he is doing. These debug commands, Macros, can be executed with a single keystroke. In addition, it is possible to pass parameters to the macros to make them more general purpose. The macro names can be saved and loaded for future debugging sessions.

The user can program PROBE to execute a macro command when a breakpoint occurs. This lets the programmer set up complex tests which display the information he wants want to see after each breakpoint or guide the PROBE to do additional testing of the target.

For more information on PROBE, call toll free 1-800-4-GESPAC (in Arizona 1-602-962-5559), or write to GESPAC, 50 W. Hoover Ave., Mesa, AZ 85210.

MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

Richard Mirabello
Sony Microsystems
(415) 965-4492

Dean Mosley
Microprocessor Products Group
(512) 440-2839

MOTOROLA INCREASES SPEED OF 68030 TO 33 MHz

Speed Increase Makes 030 Fastest General-Purpose 32-Bit Chip Available

AUSTIN, Texas, June 1, 1988—Motorola today announced an enhancement to its 32-bit 68000 family with the development of a 33 MHz 68030 (030) microprocessor. The new chip is the fastest clock-speed, general-purpose 32-bit microprocessor on the market. In a separate related announcement, Hewlett-Packard announced that it will incorporate the 33 MHz 030 in a high-end model of their Series 9000 workstation line.

The 33 MHz 030 follows the April 1988 announcement of the 25 MHz 030, and the October 1987 introduction of the 030 microprocessor at 20 MHz. Motorola's 68000 family, a compatible line of microprocessors that has generated the world's largest 32-bit software and hardware base, includes the 68000, 68010, 68020, 68030 and future 68040. The series is available at a range of speeds from 8 MHz to 33 MHz.

"The ability of the 030 to move to this high clock speed is a tribute to its design and to our track record in manufacturing," said Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group (Austin, Texas). "The 030's compatibility with preceding 68000-family processors will allow our customers to easily add performance to their existing systems."

HEWLETT-PACKARD AND SONY INTRODUCE HIGH-END WORKSTATIONS BASED ON MOTOROLA'S 68030

Widely Installed 68000 Base Continues to Support New Systems

AUSTIN, Texas, June 1, 1988—Motorola today announced that two companies, Hewlett-Packard and Sony Microsystems, will incorporate the 68030 microprocessor into their respective workstation product lines. Both Hewlett-Packard's Model 360 and Sony's NEWS 1850 incorporate a 25 MHz 68030 as a central processing unit and a 25 MHz 68882 floating-point coprocessor for mathematical operations. In addition, Sony's workstation uses a second 68030 as an embedded controller (I/O processor).

The 68000 line currently has four microprocessor members— the 68000, 68010, 68020 and 68030. Motorola has announced development of a next-generation microprocessor, the 68040. In total, more than 15 million chips from the 68000 family power a variety of applications including supercomputers, engineering workstations, business computers and embedded control devices. All generations of the 68000 are compatible with each other.

"The 68000 processor family will continue to drive much of the workstation market," said Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group (Austin, Texas). "We are dedicated to working closely with high-quality system vendors to ensure the success of 68000-based systems in the market."

Hewlett-Packard currently has more than 250,000 Motorola-based systems installed worldwide. Among these is the HP 9000 workstation series that includes the 68030-based Model 360. The HP 9000 series addresses mechanical and electrical design, test and measurement, and general scientific/technical computation. In continuing its commitment to the 68000 family, HP also announced plans to introduce a workstation based on a 33 MHz version of the 68030 later this year.

Sony's NEWS 1800 technical workstation series, based on the 68000 family, is the market leader in Japan and has more than 300 software application packages ported to it. The 68030-based NEWS 1850 is the most recent addition to the series. The NEWS 1800 Series is used in electrical and mechanical design automation applications, particularly computer-aided design, manufacturing and engineering; and computer-aided software engineering.

Motorola's $2.2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola Inc. It is the largest and broadest supplier of semiconductors in North America with a balanced portfolio of over 50,000 devices.

Classifieds

As S bmitted - No Guarantees

For Sale: 6809 Microprocessor; 80 megabyte hard drive; floppy disk drive; tape backup; 8212 terminals; printer; uniflex operating system. Cabinetry and software available. $15,500.00 Contact Cal Mueller, Boelter Industries, Inc. Phone (507) 452-2315

•••

MUSTANG-020 16Mhz with 68881. OS9 Professional Package & C $3000. Call Tom (615)842-4600.

AT&T 7300 UNIX PC, UNIX V OS, 1MB Memory, 20 MB Hard Disk, 5" Drive, Internal Modem, Mouse. Best Offer Gets It.

S+System with Cabinet, 20 Meg Hard Disk & 8" Disk Drive with DMAF3 Controller Board. 1-X12 Terminal $4800.

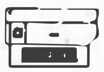
HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.
3-Dual 8" drive enclosure with power supply. New in box. $125 each.
5-Siemens 8" Disk Drive, $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS. $495.
QUME QVT-102 terminal, like new, amber screen $250. or best offer.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09CPU Card- $900 complete.
Tom (615) 842-4600 M-F 9AM to 5PM EST

•••

TWO SK*DOS Systems - (1) PT-69 6809 2-80T & 1-40T floppies, Stylo, Dataman, Dynacalc, Basic, C. $400 - (2) PT68K-1 Indentical to newest Mustang 08/A. 12MHZ 68008 768K-80T floppies &1-25M hard disk. $1200. Also available with either system: Qume QVT-108 terminal 14" amber $150, TI-855 printer with 64K external buffer and extra font cartridges $350. Package Deal: PT-69, terminal and printer for $750.
Marty Bose (415) 351-7297



NEW!

OmegaSoft Pascal for the 68020/68881

P20K is a Pascal package that will generate code for all of the 68000 series processors, including the 68881 coprocessor. P20K will run on any 68000 series computer running the OS-9/68000 (Microware) or PDOS (Eyring Research) operating systems with 512K or more free memory.

The base package (P20K-B) includes the Compiler, Relocatable Macro Assembler, Linking Loader, Screen Editor, Pascal Shell, Linkage Creator, Host Debugger, Configuration manager, Installation program, and Patch utility. A new feature in this compiler is the ability to either link in the parts of the runtime needed by the program, or to use trap handlers for runtime access, to share the runtime library between programs. Complete operating system interface is also included using pascal procedures and functions. The host debugger allows debugging at both the Pascal and assembly language levels of programs that run on the host operating system. Price for the base package is $575.

The runtime source code option (P20K-R) is available for $100 and includes source code for the operating system interface routines as well as pascal runtime.

The Utility source option (P20K-S) is available for $275 and includes source code for the Screen Editor, Pascal Shell, Host Debugger, Patch utility, and Configuration manager.

The Target debugger option (P20K-T) is $225 and includes object and source code. This program allows Pascal level and assembly level debugging in a system without operating system, by using a serial link connected to the host computer.

Prices do not include shipping charges. Master-Card and Visa accepted. OmegaSoft is a registered trademark of Certified Software Corporation.

Gespac SA, 3, Chemin des Aulx, CH-1228,
Geneva/Plan-les-Ouates, Switzerland
TEL 022-713400, TLX 429989

Elsoft AG, Zelgweg 12, CH-5405 Baden-Dättwil,
Switzerland, TEL 056-833377, TLX 828275

RCS Microsystems Ltd., 141 Uxbridge Road,
Hampton Hill, Middlesex, England
TEL 01-9792204, TLX 8951470

Byte Studio Borken, Butenwall 14, D-4280 Borken,
West Germany.
TEL 02861-2147, TLX 813343

Eltec Elektronik GmbH, Galileo-Galilei-Straße,
6500 Mainz 42, Postfach 65, West Germany
TEL 06131-50031, TLX 4187273

PEP Elektronik Systeme GmbH, Am Klosterwald 4
D-8950 Kaufbeuren, West Germany
TEL 08341-8974, TLX 541233

CERTIFIED
SOFTWARE
CORPORATION

616 CAMINO CABALLO, NIPOMO, CA 93444 USA
TEL (805) 929-1395 TELEX 467013
FAX (805) 929-1395 (MID-8AM)

FLEX™/SK-DOS™/MS-DOS™

Transfer Utilities

For 68XXX and CoCo* OS-9™ Systems

Now READ - WRITE - DIR - DUMP - EXPLORE
FLEX, SK-DOS & MS-DOS Disk

These Utilities come with a rich set of options allowing the transfer of text type files from/to FLEX & MS-DOS disks.

*CoCo systems require the D.P. Johnson SDISK utilities and OS-9 and two drives of which one must be a "host" floppy.

CoCo Version: $69.95

68XXX Version $99.95

S.E. Media ←→ **PO Box 849, Hixson,TN 37343**

615 842-6809

MC/Visa

K-BASIC™

The Only 6809 BASIC to Binary Compiler for OS-9
FLEX or SK*DOS
Even runs on the 68XXX SK*DOS Systems*

Hundreds Sold at Suggested Retail:

~~$199.00~~

- 6809 - OS-9™ users can now transfer their FLEX™ Extended BASIC (XBASIC) source files to OS-9, compile with the OS-9 version and run them as any other OS-9 binary "CMD" program. Much faster than BASIC programs.

- 6809 - FLEX users can compiler their BASIC source files to a regular FLEX ".CMD" file. Much faster execution.

- 68XXX - SK*DOS™ users running on 68XXX systems (such as the Mustang-08/A) can continue to execute their 6809 FLEX BASIC and compiled programs while getting things ported over to the 68XXX. SK*DOS allows 6809 programs to run in emulation mode. This is the only system we know of that will run both 6809 & 68XXX binary files.

K-BASIC is a true compiler. Compiling BASIC 6809 programs to binary command type programs. The savings in RAM needed and the increased speed of binary execution makes this a must for the serious user. And the price is now RIGHT!

Don't get caught up In the *"Learn a New Language"* syndrome - Write Your Program In BASIC, Debug It In BASIC and Then Compile It to a .CMD Binary File.

For a LIMITED time save over 65%... This sale will not be repeated after it's over! *

SALE SPECIAL:

$69.95

SPECIAL

Thank-You-Sale

Only From:

S.E. Media™

CPI

5900 Cassandra Smith Rd.
Hixson, Tn 37343
Telephone 615 842-6809
Telex 510 600-6630

A Division of Computer Publishing Inc.
Over 1,200 Titles - 6800-6809-68000

* K-BASIC will run under 68XXX SK*DOS in emulation mode for the 6809.
Price subject to change without notice.

ATARI & AMIGA CALL

As most of you know, we are very sensitive to your wishes, as concerns the contents of these pages. One of the things that many of you have repeatedly written or called about is coverage for the **Atari & Amiga™** series of 68000 computers.

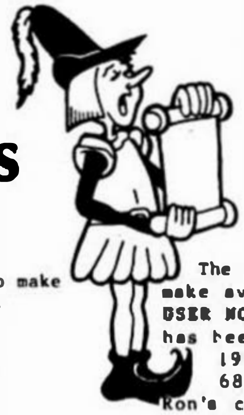
Actually we haven't been too keen on those systems due to a lack of serious software. They were mainly expensive "game-toy" systems. However, recently we are seeing more and more honest-to-goodness serious software for the Atari & Amiga machines. That makes a difference. I feel that we are ready to start some serious looking into a section for the Atari & Amiga computers. Especially so since OS-9 is now running on the Atari (review copy on the way for evaluation and report to you) and rumored for the Amiga. Many of you are doing all kinds of interesting things on these systems. By sharing we all benefit.

This I must stress - *Input from you on the Atari & Amiga.* As most of you are aware, we are a "contributor supported" magazine. That means that YOU have to do your part. Which is the way it has been for over 10 years. We need articles, technical, reviews of hardware and software, programming (all languages) and the many other facets of support that we have pursued for these many years. Also I will need several to volunteer to do regular columns on the Atari & Amiga systems. Without constant input we can't make it fly! So, if you do your part, we certainly will do ours. How about it, drop me a line or give me a phone call and I will get additional information right back to you. We need your input and support if this is to succeed!

DMW

THE 6800-6809 BOOKS

..HEAR YE.....HEAR



OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO C1	File load program to offset memory — ASM PIC
MEMOVE C1	Memory move program — ASM PIC
DUMP C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM C2	Modem input to disk (or other port input to disk) — ASM
M C2	Output a file to modem (or another port) — ASM
PRINT C3	Parallel (enhanced) printer driver — ASM
MODEM C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG C1	Scientific math routines — PASCAL
U C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT C4	Parallel printer driver, without PFLAG — ASM
SET C5	Set printer modes — ASM
SETBAS1 C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set

Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail

If paying by check - Please allow 4-6 weeks delivery

*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

!!! Subscribe Now !!!
68 MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Mastercard □ VISA □

Card #______________________ Exp. Date________

For 1 Year ____ 2 Years ____ 3 Years ____

Enclosed: $____________

Name ______________________________

Street ______________________________

City ______________ **State** ____ **Zip** _______

Country ______________________________

My Computer Is: ______________

Subscription Rates

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

•Foreign Surface: Add $12.00 per Year to USA Price.

•Foreign Airmail: Add $48.00 per Year to USA Price.

•Canada & Mexico: Add $9.50 per Year to USA Price.

•U.S. Currency Cash or Check Drawn on a USA Bank !

68 Micro Journal
5900 Cassandra Smith Rd.
POB 849
Hixson, TN 37343

VISA mastercard

Telephone 615 842-4600
Telex 510 600-6630

Reader Service Disks

Disk- 1	Filesort, Minicat, Minicopy,Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk- 2	Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk- 3	Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk- 4	Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6	**Purchase Order, Index (Disk file indx).
Disk- 7	Linking Loader, Rload, Hardness.
Disk- 8	Cnest, Lanpher (May 82).
Disk- 9	Datcopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Dissembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initmf68, Testmf68,*Cleanup,*Dskalign,Help,Date.Txt.
Disk-14	*Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven, Language Recognition Utility, Anderson March 1986.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**Star Trek
Disk-30	Simple Winchester, Dec.'86 Green.
Disk-31	*** Read/Write MS/PC-DOS (SK*DOS)
Disk-32	Heir-UNIX Type upgrade - 68MJ 2/87
Disk-33	Build the GT-4 Terminal - 68MJ 11/87 Condon.
Disk-34	FLEX 6809 Diagnostics, Disk Drive Test, ROM Test, RAM Test - 68MJ 4/88Korpi.

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

• Denotes 6800 - •• Denotes BASIC
••• Denotes 68000 - 6809 no indicator.

VISA MasterCard

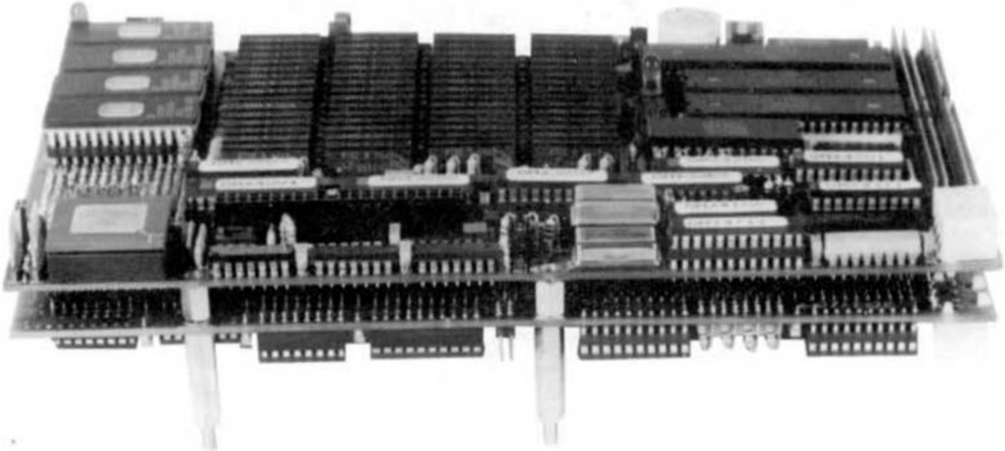
8" disk $19.50
5" disk $16.95

Shipping & Handling -U.S.A. Add: - $3.50
Overseas add: $4.50 Surface - $7.00 Airmail

68 MICRO JOURNAL

5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4600 - Telex 510 600-6630

NOW THE GMX MICRO-20 HAS A TWIN —
Announcing The GMX TWINGLE-20
68020 TWIN BOARD COMPUTER WITH MMU



All the features of the GMX Micro-20, PLUS —

- **2 Megabytes additional RAM — for a total of 4 Megabytes of RAM**
- **8 more Serial ports — for a total of 12, and expandable up to 44.**
- **MEMORY MANAGEMENT UNIT**

The GMX TWINGLE-20 consists of 2 boards. One of the boards is the same as the Micro-20, except for the 68020 processor which is on the MMU board. It uses the same I/O expansion interface, serial adapter boards, and mounting holes as the GMX Micro-20, making it easy to upgrade existing systems. Any of the currently available GMX Micro-20 I/O expansion boards can be used to provide additional I/O capability. Expansion possibilities include additional serial ports (up to 44 ports total), additional parallel ports, and local area networking of up to 255 GMX Micro-20s and/or TWINGLE-20s.

The MMU board contains the additional 2 Megabytes of RAM, 8 serial ports with 2 connectors for the SAB 4 port adaptor cards, and the MMU hardware. The MMU is a proprietary high-speed design that fully supports virtual memory. The system RAM normally operates with only 1 wait-state, regardless of processor speed. An additional wait-state is needed only when program flow crosses a 4K boundary. The MMU can be configured for any one of four different maps, ranging from 8 tasks with 8 megabytes of virtual address space each, to 64 tasks of 1 megabyte each. The MMU can be disabled for applications that do not use hardware memory management.

The TWINGLE-20 two board set can occupy the same space as a half-height 5.25" disk drive. It is available in 12.5, 16.67 or 20 MHz. versions, and with or without the 68881 FPC.

SPECIFICATIONS

Size: 8.8 × 5.75 × 1.4 inches.

Power Requirements: +5VDC @ 8.3A typical (20MHz, with 68881).
The TWINGLE-20 itself does not require a +12V supply. +12V supply requirements, if any, are determined by the serial adapter boards and any I/O expansion boards powered through the I/O Expansion Interface.

SOFTWARE INCLUDED:

An enhanced version of 020Bug with diagnostics for the MMU and the additional RAM and serial ports.

OPTIONAL SOFTWARE:

UniFLEX VM, Virtual Memory version of the UniFLEX operating system which includes all of the features of the GMX Micro-20 version, plus full MMU support.

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

- Compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 8 Megabytes of Virtual Memory per user.

All the optional languages and software that run under UniFLEX for the Micro-20 are also available for the TWINGLE-20.

OS-9 Users can take advantage of the additional RAM and serial ports on the TWINGLE-20. It does not presently support the MMU.

GMX INC 1337 W. 37th Place, Chicago, IL 60609 (312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352

For Ordering Call

(615)842-4600

FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+ ™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.⁰⁰ **Only $79.⁹⁵**

STAR-DOS PLUS+
- ● Functions Same as FLEX
- ● Reads - writes FLEX Disks
- ● Run FLEX Programs
- ● Just type: Run "STAR-DOS"
- ● Over 300 utilities & programs to choose from.

$34.⁹⁵

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.⁹⁵

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

TSC Editor
Reg $50.00
NOW $35.00

- + Read-Write-Dir RS Disk
- + Run RS Basic from Both
- + More Free Utilities
- + External Terminal Program
- + Test Disk Program
- + Disk Examine & Repair Program
- + Memory Examine Program
- + Many Many More!!!

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JPD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95

* Specify What CONTROLLER You Went J&M, or RADIO SHACK

THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JPD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$125.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

☎
(615)842-4600
For Ordering
Telex 5106006630

An Ace of a System in Spades! The New

MUSTANG-08/A™

Now with 4 serial ports standard & speed increase to 12 Mhz CPU + on board battery backup and includes the PROFESSIONAL OS-9 package - including the $500.00 OS-9 C compiler! This offer won't last forever!

NOT 128K, *NOT* 512K

FULL 768K No Wait RAM

The MUSTANG-08™ system took every hand from all other 68008 sysems we tested, running OS-9 68K!

The MUSTANG-08 includes OS9-68K™ and/or Peter Stark's SK*DOS™. SK*DOS is a single user, single tasking system that takes up where *FLEX™ left off. SK*DOS is actually a 68XXX FLEX type system *(Not a TSC product.)*

The OS-9 68K system is a full blown multi-user, multi-tasking 68XXX system. All the popular 68000 OS-9 software runs. It is a speed *whiz on disk I/O*. Fact is: the MUSTANG-08 is faster on disk access than some other 68XXX systems are on memory cache access. Now, that is fast! And that is just a small part of the story! See benchmarks.

SALE!

System includes OS-9 68K or SK*DOS - Your Choice

Specifications:

CPU	MC68008	12 Mhz
RAM	768K	256K Chips
	No Wait States	
PORTS	4 - RS232	MC68681 DUART
	2 - 8 bit Parallel	MC6821 PIA
CLOCK	MK48T02	Real Time Clock Bat. B/U
EPROM	16K, 32K or 64K	Selectable
FLOPPY	WD1772	5 1/4 Drives
HARD DISK	Interface Port	WD1002 Board

Now more serial ports - faster CPU Battery B/U - and $850.00 OS-9 Professional with C compiler included!

***$400.00** See Mustang-02 Ad - page 5 for trade-in details

Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

MUSTANG-08

LOOK

	Seconds	32 bit Integer	Register Long
Other 68008 8 Mhz OS-9	68K...	18.0...	9.0
MUSTANG-08 10 Mhz OS-9	68K....	9.8...	6.3

C Benchmark Loop

```
Main()
{
        /* int i; */
        register long i;
        for (i=0; i < 999999; ++i);
}
```

Now even faster! with 12 Mhz CPU

C Compile times: OS-9 68K Hard Disk

MUSTANG-08	8 Mhz CPU	0 min - 32 sec
Other popular 68008 system		1 min - 05 sec
MUSTANG-020		0 min - 21 sec

25 Megabyte Hard Disk System

$2,398.90

Complete with PROFESSIONAL OS-9 includes the $500.00 C compiler, PC style cabinet, heavy duty power supply, 5" DDDS 80 track floppy, 25 MegByte Hard Disk - Ready to Run

Unlike other 68008 systems there are several significant differences. The MUSTANG-08 is a full 12 Megahertz system. The RAM uses NO wait states, this means full bore MUSTANG type performance.

Also, allowing for addressable ROM/PROM the RAM is the maximum allowed for a 68008. The 68008 can only address a total of 1 Megabytes of RAM. The design allows all the RAM space (for all practical purposes) to be utilized. What is not available to the user is required and reserved for the system.

A RAM disk of 480K can be easily configured, leaving 288K free for program/system RAM space. The RAM DISK can be configured to any size your application requires (system must have 128K in addition to its other requirements). Leaving the remainder of the original 768K for program use. Sufficient source included (drivers, etc.)

FLEX is a trademark of TSC   MUSTANG-08 is a trademark of CPI

Data-Comp Division

A Decade of Quality Service"
Systems World-Wide

Computer Publishing, Inc.   5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630   Hixson, Tn 37343

* Those with SWTPC hi-density FLEX 5" - Call for special info.